TELECOMUNICACIÓN

Campus Sur

POLITÉCNICA

PROYECTO FIN DE GRADO

TÍTULO: SISTEMA DE TELEMETRÍA PARA UN VEHÍCULO DE FÓRMULA S.A.E.

AUTOR: JAIME IGLESIAS HERNÁNDEZ

TUTOR (o Director en su caso): EDUARDO BARRERA

DEPARTAMENTO: SEC

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

V°B°

Miembros del Tribunal Calificador:

PRESIDENTE: EDUARDO NOGUEIRA DÍAZ

TUTOR: EDUARDO BARRERA

SECRETARIO: SERGIO LÓPEZ GREGORIO

Fecha de lectura: 24 **de** septiembre **de** 2013

Calificación:

El Secretario,

ÍNDICE

Índice de tablas

RESUMEN

En este proyecto, se ha desarrollado una aplicación electrónica para un coche de competición, en concreto para la fórmula SAE (Society of Automotive Engineers), una competición universitaria en la que cada equipo, formado por estudiantes, debe diseñar, construir y probar un prototipo basándose en una serie de reglas. El objetivo final de la competición es proporcionar a los estudiantes el conocimiento práctico necesario para su futura labor profesional, del cual se pensaba que los estudiantes adolecían al acabar sus estudios universitarios cuando se creó esta competición.

La aplicación desarrollada en este proyecto consiste en un sistema de telemetría, utilizado para transmitir los datos proporcionados por los sensores del vehículo a través de un sistema de radiofrecuencia, de manera que se pueda estudiar el comportamiento del coche durante los ensayos a la vez que el coche está rodando y así no depender de un sistema de adquisición de datos del que había que descargarse la información una vez finalizada la sesión de ensayo, como había que hacer hasta el momento.

Para la implementación del proyecto, se ha utilizado un kit de desarrollo (Xbee Pro 868) que incluye dos módulos de radio, dos placas de desarrollo, dos cables USB y una antena, el cual ha permitido desarrollar la parte de radio del proyecto.

Para transmitir los datos proporcionados por la centralita del vehículo, la cual recoge la información de todos los sensores presentes en el vehículo, se han desarrollado dos placas de circuito impreso. La primera de ellas tiene como elemento principal un microprocesador PIC de la marca Microchip (PIC24HJ64GP502), que recoge los datos proporcionados por la centralita del vehículo a través de su bus CAN de comunicaciones.

La segunda placa de circuito impreso tiene como elemento fundamental el transmisor de radio. Dicho transmisor está conectado al microprocesador de la otra placa a través de línea serie.

Como receptor de radio se ha utilizado una de las placas de prueba que integraba el kit de desarrollo Xbee Pro 868, la cual recoge los datos que han sido enviados vía radio y los manda a su vez a través de USB a un ordenador donde son monitorizados. Hasta aquí la parte hardware del sistema.

En cuanto a la parte software, ha habido que desarrollar una aplicación en lenguaje C, que ejecuta el microprocesador PIC, que se encarga de recoger los datos enviados por la centralita a través del bus CAN (Controller Area Network) y transmitirlos a través de línea serie al chip de radio. Por último, para la monitorización de los datos se han desarrollado dos aplicaciones en LabVIEW, una que recoge los datos a través de USB, los muestra en pantalla y los guarda en un fichero y otra que lee los datos del fichero y los representa gráficamente para permitir un estudio más detallado del comportamiento del vehículo.

ABSTRACT

In this project, an electronic application has been developed for a race car – Formula SAE car-. Formula SAE is a university championship in which each team, made up of students, should design, construct and test a prototype within certain rules. The final goal of the competition is to enhance the practical knowledge of the students, which was thougth to be poor at the time the competition was created.

The application developed in this project consists of a telemetry system, employed to transmit the data provided by the car's sensors through a radio frequency system, so that it could be possible to study the behaviour of the vehicle during tests and do not depend on a datalogger system as it occurred until now.

To carry out the radio module of the project, a Xbee Pro 868 development kit has been used, which includes two radio modules, two development boards, two USB cables and an antenna.

To transmit the data provided by the ECU (Engine Control Unit) of the vehicle, which receives information from all the sensors the vehicle has, two printed circuit boards have been built. One of them has a PIC microprocessor of Microchip (PIC24HJ64GP502) which receives the data coming from CAN bus of the ECU.

Tha main element of the other printed circuit board is the radio transmitter. This chip receives the data from the microprocessor through its serial line.

The development board of the Xbee Pro 868 has been used as receiver. When data arrives to the receiver, it transmits them to a computer through USB where the data are displayed. All this composes the hardware of the system.

Regarding the software, a C coded application has been developed. This application is executed by the microprocessor and its function is to receive the data from the bus CAN (Controller Area Network) and send them to the radio transmitter through the microprocessor's serial line. To show the data on the computer, two LabVIEW applications has been developed. The first one receives the data through the USB port, displays them on the screen and save them to a file and the second one reads the data from the file while represents them graphically to allow studying the behaviour of the car on track.

INTRODUCCIÓN

El proyecto que se ha desarrollado es una aplicación electrónica para un coche de carreras tipo fórmula, que compite en la fórmula SAE (Society of Automotive Engineers). La fórmula SAE es una competición de diseño para estudiantes universitarios organizada por la Society of Automotive Engineers. La primera edición de la competición se celebró en 1978 y originalmente se llamó Mini Indy. Su objetivo fundamental es implicar a jóvenes ingenieros en el diseño, construcción y puesta a punto de un vehículo tipo fórmula. El ámbito de la competición se extiende desde la evaluación del diseño, habilidades de marketing, costes y rendimiento ante varios ensayos dinámicos, hasta la realización de una carrera de 22km en un circuito de máxima exigencia.

En la Fórmula SAE, participan 200 equipos de las más prestigiosas universidades de todo el mundo. La competición y sus carreras tienen lugar todos los años en varios países de los cinco continentes. El primer equipo español en participar en la Fórmula SAE fue el promovido por el **INSIA** (Instituto Universitario de Investigación del Automóvil): el **UPM Racing**. Este equipo cuenta con el patrocinio, además del INSIA, de diversas empresas privadas del área de la automoción. Como recurso humano se nutre de estudiantes de **la E.T.S.I.I** (Escuela Técnica Superior de Ingenieros Industriales), alumnos del **Máster de Ingeniería de Automoción**, alumnos de la escuela de **Ingeniería Técnica Aeronáutica** y de la **EUITT** (Escuela Universitaria de Ingeniería Técnica de Telecomunicaciones).

Este proyecto ha consistido en realizar un sistema de telemetría con el cual se puedan monitorizar distintos parámetros del comportamiento del coche y del motor como pueden ser velocidad, temperaturas de aceite, agua o aire, tensión de batería etcétera.

La normativa de la competición obliga a realizar un coche nuevo cada año y a parte de esto, cada año se trata de mejorar el coche por lo que se introducen piezas nuevas y se modifican otras que necesitan ser probadas en pista. Con la telemetría se pretende facilitar el proceso de validación de dichas piezas y ayudar a detectar de manera rápida errores que pueden ser críticos para el motor como un excesivo calentamiento o caídas en la presión de aceite.

El motor del vehículo es controlado por una centralita o ECU (Engine Control Unit) que, aparte de gestionar el proceso de inyección y encendido recibe la información de una serie de sensores que incorpora el motor. Toda esta información es transmitida por la ECU a través de un bus de comunicaciones, llamado bus CAN.

El sistema desarrollado se encarga de recibir la información que proporciona el bus CAN, procesarla con la ayuda de un microprocesador y transmitirla a través de un enlace de radio a un punto remoto donde los datos se visualizan en un ordenador.

Para llevar a cabo este proceso ha habido que desarrollar dos placas de circuito impreso y se han utilizado dos kits de desarrollo, uno para la parte de radio y otro para desarrollar toda la parte relacionada con el microprocesador, lo que compone el hardware del proyecto.

Para controlar dicho hardware se ha desarrollado una aplicación en lenguaje C, la cual realiza todo el procesamiento de datos en el microprocesador y para visualizar los datos que llegan al receptor de radio se ha programado una aplicación en LabVIEW.

Estas dos partes se explican detalladamente en los capítulos dos y tres. A continuación y como introducción al capítulo uno, se expone brevemente como y para qué surge el bus CAN de datos.

ANTECEDENTES O MARCO TECNOLÓGICO

[1] Finalizando el año 1989, la industria automovilística tenía fundamentalmente tres problemas:

El primero era la demanda de mayor comodidad en los automóviles particulares, tales como ventanillas accionadas eléctricamente, ajuste de asientos y espejos, equipos de audio y vídeo, etc.

El segundo, era la seguridad en el vehículo, sistemas de inmovilización, cierre centralizado de puertas, sistemas antibloqueo de frenos (ABS), protección contra accidentes (AIR BAG), control de velocidad, etc.

El tercer problema era el relacionado con el consumo, el rendimiento y la contaminación.

Los tres problemas eran encarados por medio de control electrónico, al principio, con una única unidad electrónica de control, y luego con el agregado de otras. Entonces, surgió la necesidad de establecer una adecuada comunicación entre las distintas unidades electrónicas de control de dichos procesos. Una estimación futura afirmaba que para el año 2005 los automóviles podían contar con hasta 100 microprocesadores especializados. El cableado necesario para establecer la comunicación entre todos estos microprocesadores sería muy extenso y además requeriría distintos tipos de cable. Para solucionar este problema, la industria del automóvil empleó buses de comunicaciones. Lamentablemente, los distintos fabricantes de automóviles desarrollaron sus propios buses y cada sistema era incompatible con los otros. En los primeros años de la década de los 90, surgieron los bus CAN, VAN, J1850SCP Y J1850DLC, posteriormente abandonados a favor del CAN, que hoy todavía se sigue usando como líder mundial en bus para automóviles.

Un sistema CAN (Controller Area Network) es un sencillo bus serie diferencial de dos cables, ideal para reducir el cableado. CAN significa que las unidades de control están interconectadas e intercambian datos entre sí.

El bus CAN de datos representa un modo de transmitir los datos entre las distintas unidades de control. Comunica las diferentes unidades de control en un sistema global interconectado.

Ventajas del bus CAN de datos:

- Si el protocolo de datos ha de ser ampliado con información suplementaria, solamente se necesitan modificaciones en el software.
- Un bajo porcentaje de errores mediante una verificación continúa de la información transmitida, de parte de las unidades de control, y mediante protecciones adicionales en los protocolos de datos.
- Menos sensores y cables de señal gracias al uso múltiple de una misma señal de sensores.
- Es posible una transmisión de datos muy rápida entre las unidades de control, hasta 1 Mbps.
- Más espacio disponible, mediante unidades de control más pequeñas y conectores más compactos para las unidades de control.
- El bus CAN de datos está normalizado a nivel mundial. Por ese motivo, también las unidades de control de diferentes fabricantes pueden intercambiar datos.

CAPITULO 1: LAS REDES CAN

1.1 PRINCIPIO DE FUNCIONAMIENTO DEL BUS CAN

[1] La transmisión de datos a través del bus CAN funciona de un modo parecido al de una conferencia telefónica. Una unidad de control introduce sus datos en la red, mientras que las demás unidades de control "coescuchan" esos datos. Para ciertas unidades de control, estos datos resultan interesantes, en virtud de lo cual los utilizan, mientras que a las demás unidades de control pueden no interesarles esos datos específicos y por tanto los desechan.

1.2 COMPONENTES DE UN BUS CAN

Un bus CAN de datos consta de un controlador, un transceptor, dos elementos finales del bus y dos cables para la transmisión de datos, como se muestra en la Figura 1.1. Con excepción de los cables del bus, todos los componentes están alojados en las unidades de control.

- El controlador CAN: recibe del microprocesador, en la unidad de control, los datos que han de ser transmitidos, los acondiciona y los pasa al transceptor CAN. Asimismo, recibe los datos procedentes del transceptor CAN, los acondiciona y los pasa al microprocesador en la unidad de control.

- El transceptor CAN: es un transmisor y un receptor. Transforma los datos del controlador CAN en señales eléctricas y las transmite sobre los cables del bus CAN. Asimismo recibe los datos y los transforma para el controlador CAN.

- Elemento final del bus de datos: es una resistencia. Evita que los datos transmitidos sean devueltos en forma de eco de los extremos de los cables y que se falsifiquen los datos.

- Cables del bus de datos: funcionan de forma bidireccional y sirven para la transmisión de los datos. Se denominan CAN-High (señales de nivel lógico alto) y CAN-Low (señales de nivel lógico bajo)

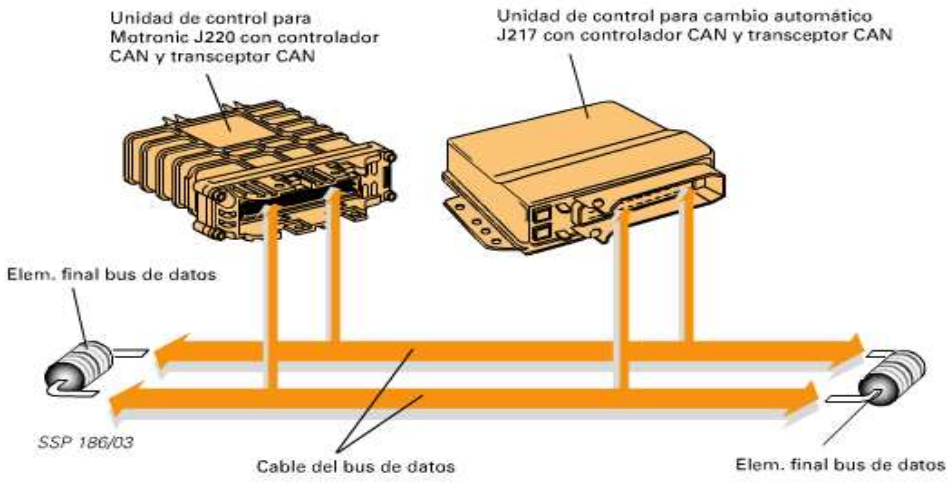


Figura 1.1 – Componentes de un bus CAN [1]

1.3 DESARROLLO DE UN CICLO DE TRANSMISIÓN DE DATOS

Como se muestra en la Figura 1.2, los pasos que se siguen en un ciclo de transmisión son los siguientes:

- Proveer datos: la unidad de control provee los datos al controlador CAN, para su transmisión.
- Transmitir datos: el transceptor CAN recibe los datos del controlador CAN, los transforma en señales eléctricas y los transmite.
- Recibir datos: todas las demás unidades de control que estén interconectadas a través del bus CAN se transforman en receptores.
- Revisar datos: las unidades de control revisan si necesitan los datos recibidos para la ejecución de sus funciones o si no los necesitan.
- Adoptar datos: si se trata de datos importantes, la unidad de control en cuestión los adopta y procesa; si no son importantes los desprecia.

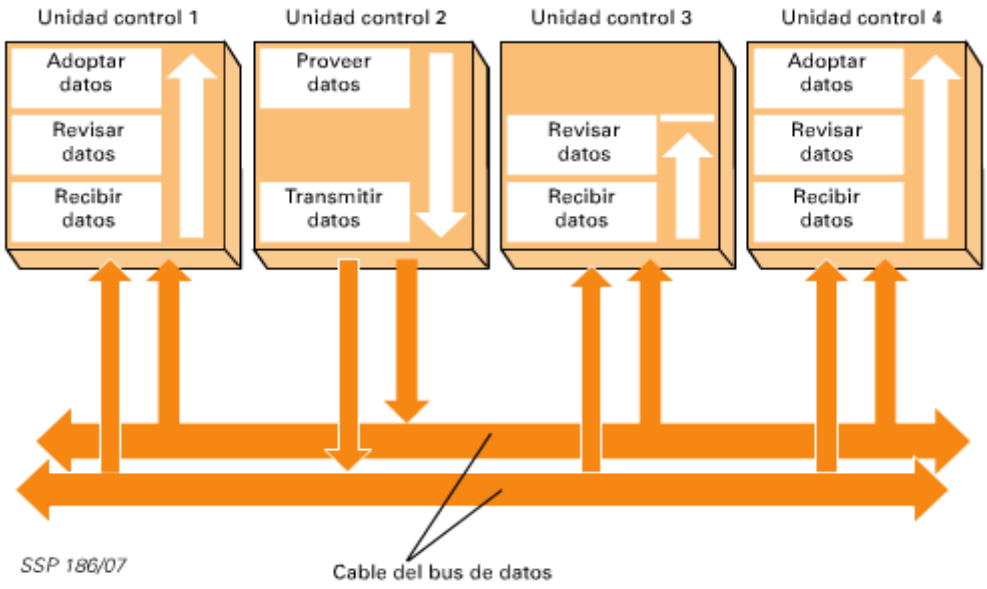


Figura 1.2 – Desarrollo de un ciclo de transmisión de datos [1]

1.4 ¿QUÉ TRANSMITE EL BUS CAN?

El protocolo del bus CAN utiliza comunicación asíncrona. La información circula desde los transmisores a los receptores en tramas de datos, las cuales están compuestas de bytes que definen el contenido de la trama de datos, como se muestra en la figura 1.3.

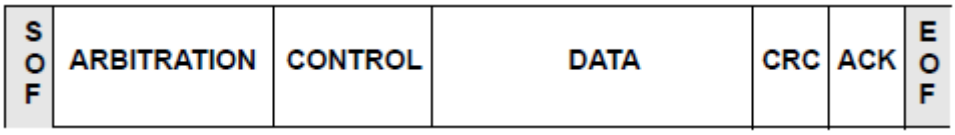


Figura 1.3 – Mensaje del bus CAN [2]

Cada trama comienza con un bit de Start-of-Frame (SOF) y termina con un campo de End-of-Frame (EOF). El bit de SOF va seguido de los campos Arbitration y Control que identifican el tipo de mensaje, el formato, la longitud y la prioridad del mismo. Esta información permite a cada nodo del bus CAN responder adecuadamente al mensaje. El campo de datos transporta el contenido del mensaje y es de un tamaño variable, pudiendo contener desde 0 hasta 8 bytes. Los campos CRC (Cyclic Redundancy Check) y ACK (Acknowledgement) proporcionan protección contra errores.

El protocolo del bus CAN soporta cinco tipos de tramas:

- Trama de datos: lleva información desde el transmisor al receptor.
- Trama remota: transmitida por un nodo al bus, para solicitar la transmisión de una trama de datos con el mismo identificador desde otro nodo.
- Trama de error: transmitida por cualquier nodo que detecta un error.
- Trama de sobrecarga: proporciona un retardo extra entre tramas sucesivas de datos o remotas.
- Espacio entre tramas: proporciona una separación entre tramas sucesivas.

La especificación 2.0B define dos formatos de datos adicionales:

- Trama de datos estándar: destinada para mensajes estándar que utilizan identificadores de 11 bits.
- Trama de datos extendida: destinada para mensajes que utilizan identificadores de 29 bits.

Existen tres versiones de las especificaciones del bus CAN:

- 2.0A: considera el identificador de 29 bits como un error.
- 2.0B Pasivo: ignora los mensajes con identificadores de 29 bits.
- 2.0B Activo: maneja tanto los identificadores de 11 bits como los de 29 bits.

1.4.1 TRAMA DE DATOS ESTÁNDAR

Las tramas de datos estándar comienzan con un bit de SOF seguido del campo Arbitration de 12 bits, tal como se muestra en la figura 1.4. El campo Arbitration se compone de un identificador de 11 bits y un bit de Petición de Transmisión Remota (RTR). El identificador define el tipo de información contenida en el mensaje y es usado por cada nodo receptor para determinar si el mensaje es de su interés o no. El bit RTR sirve para distinguir una trama de datos de una trama remota. En una trama de datos estándar, el bit RTR está inactivo (estado lógico ‘0’). Después del campo Arbitration está el campo Control, de 6 bits, el cual proporciona más información acerca del contenido del mensaje. El primer bit del campo Control, es el bit IDE (Identifier Extension), que ayuda a distinguir entre tramas de datos estándar y extendidas. Una trama de datos estándar se indica mediante un estado dominante (nivel lógico ‘0’) durante la transmisión del bit IDE. El segundo bit del campo Control es un bit reservado (RB0), el cual está a nivel bajo. Los últimos cuatro bits del campo Control representan el DLC (Data Length Code) o tamaño del campo de datos, que especifica el número de bytes de datos que contiene el mensaje.

El siguiente campo es el campo de datos. Este campo contiene los datos del mensaje. El tamaño de este campo puede variar desde 0 hasta 8 bytes. El número de bytes es configurable por el usuario.

El campo de datos va seguido del campo CRC que es una secuencia de 15 bits seguida de un bit delimitador.

El campo ACK se compone de dos bits, el ACK SLOT y el ACK DELIMITER. El transmisor envía estos dos bits en estado recesivo (nivel lógico ‘1’) y el receptor que recibe el mensaje correctamente sobrescribe el bit ACK SLOT con un bit a nivel bajo.

El último campo es el campo EOF, que consiste en 7 bits a nivel alto que indican el final del mensaje.

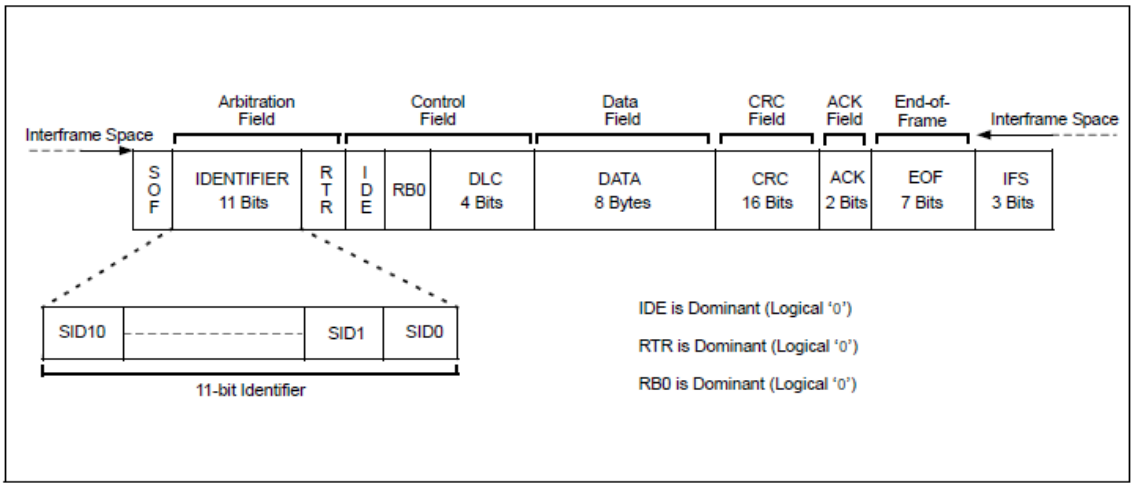


Figura 1.4- Formato de la trama de datos estándar [2]

1.4.2 TRAMA DE DATOS EXTENDIDA

La trama de datos extendida también comienza con el campo SOF y va seguido de un campo Arbitration de 31 bits, tal como se muestra en la figura 1.5. El campo Arbitration de las tramas de datos extendidas contiene un identificador de 29 bits separado en dos campos por los bits SRR (Substitute Remote Request) e IDE. El bit SRR = 1 en las tramas de datos extendidas. El bit IDE indica el tipo de trama de datos. En las tramas extendidas IDE = 1.

El campo Control en las tramas de datos extendidas es de siete bits. El primer bit es el RTR, que vale 0 en este tipo de tramas. Los siguientes dos bits, RB1 y RB0, son bits reservados y se encuentran a nivel bajo. Los últimos cuatro bits del campo Control representan el DLC o tamaño del campo de datos, que especifica el número de bytes de datos que contiene el mensaje.

Los siguientes campos de las tramas de datos extendidas son idénticos a los de las tramas estándar.

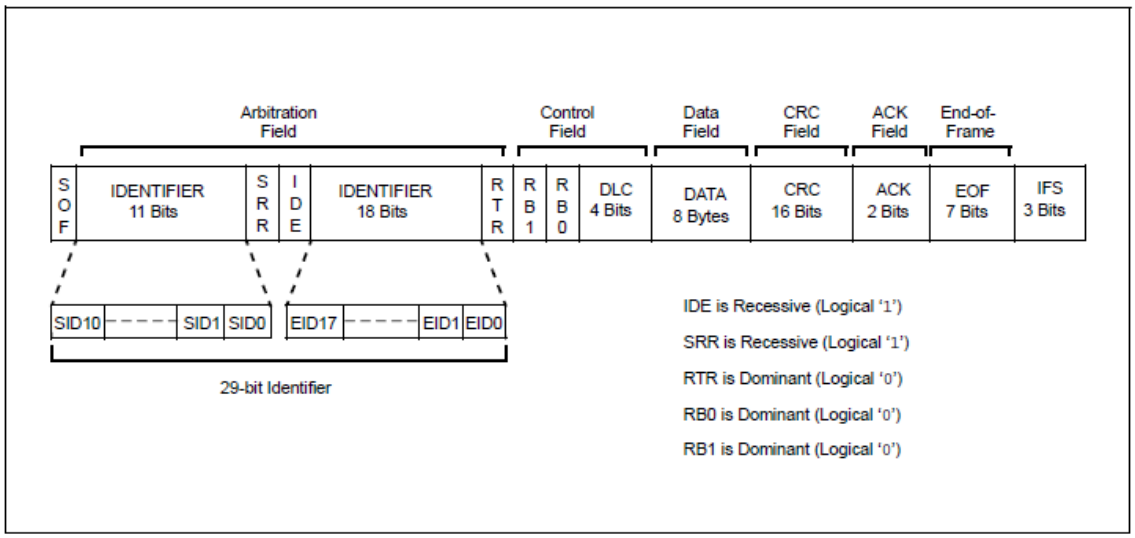


Figura 1.5 – Formato de la trama de datos extendida [2]

1.4.3 TRAMA REMOTA

Un nodo que está esperando recibir datos de otro nodo puede iniciar la transmisión enviando al nodo transmisor una trama remota. La trama remota puede estar en formato estándar (Figura 1.6) o en formato extendido (Figura 1.7).

Una trama remota es similar a una trama de datos, con las siguientes excepciones:

- El bit RTR es recesivo (RTR = 1)

- No hay campo de datos (DLC = 0)

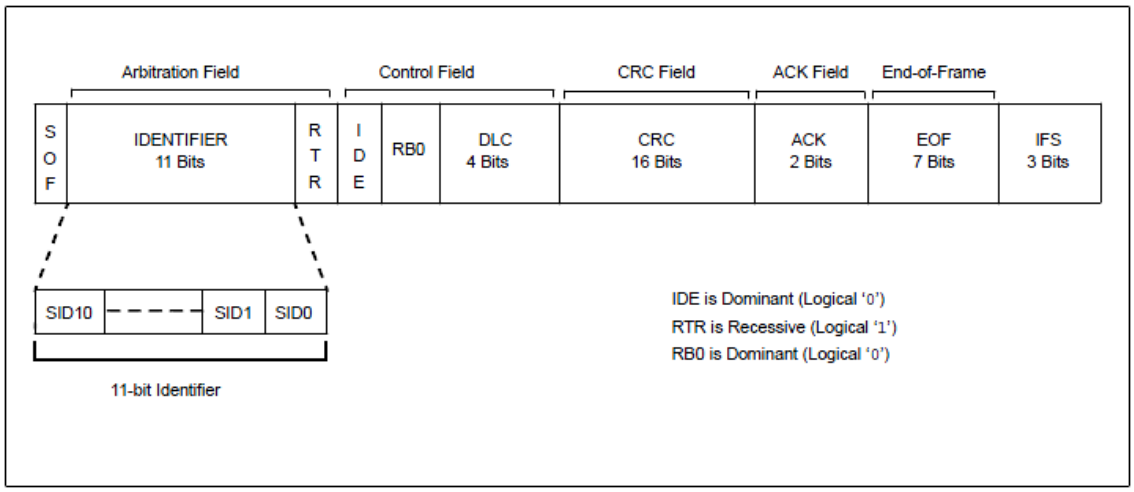


Figura 1.6 – Formato de la trama remota estándar [2]

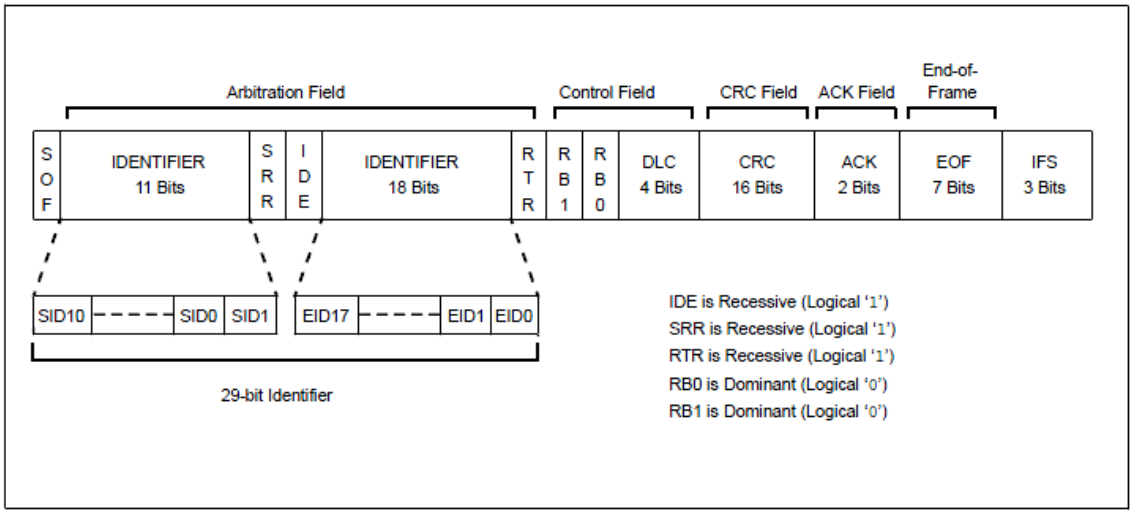


Figura 1.7 – Formato de la trama remota extendida [2]

1.4.4 TRAMA DE ERROR

Una trama de error es generada por cualquier nodo que detecte un error en el bus. Como se puede apreciar en la Figura 1.8, estas tramas se componen de dos campos: Error Flag y Error Delimiter. El campo Error Delimiter consiste en 8 bits a nivel alto y permite a los nodos del bus restablecer las comunicaciones después de un error. Hay dos tipos de campos Error Flag, dependiendo del estado del nodo que detecta el error:

- Error Flag Activo: contiene 6 bits a nivel bajo consecutivos, lo que fuerza a otros nodos de la red a generar Error Echo Flags, resultando por tanto en una serie de 6 a 12 bits a nivel bajo en el bus.
- Error Flag Pasivo: contiene 6 bits a nivel alto consecutivos, con el resultado de que a menos que el error en el bus sea detectado por el nodo transmisor, la transmisión de un flag de error pasivo no afectará a las comunicaciones de ningún otro nodo de la red.

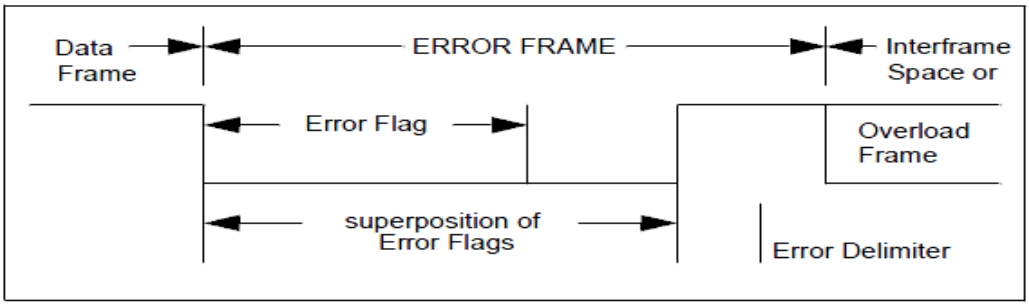


Figura 1.8 – Formato de la trama de error [3]

1.4.5 TRAMA DE SOBRECARGA

Una trama de sobrecarga puede ser generada por un nodo, tanto cuando se detecta un bit a nivel bajo durante el espacio entre tramas (Interframe Spacing), como cuando un nodo todavía no está listo para recibir el siguiente mensaje (por ejemplo, si está todavía leyendo el mensaje recibido previamente). Una trama de sobrecarga tiene el mismo formato que una trama de error con el flag de error activo, pero solo puede generarse durante el espacio entre tramas. Como se ve en la Figura 1.9, consiste en un campo Overload Flag, compuesto por 6 bits a nivel bajo, seguido de un campo Overload Delimiter, compuesto de 8 bits a nivel alto. Un nodo puede generar un máximo de dos tramas de sobrecarga seguidas para retrasar el comienzo del siguiente mensaje.

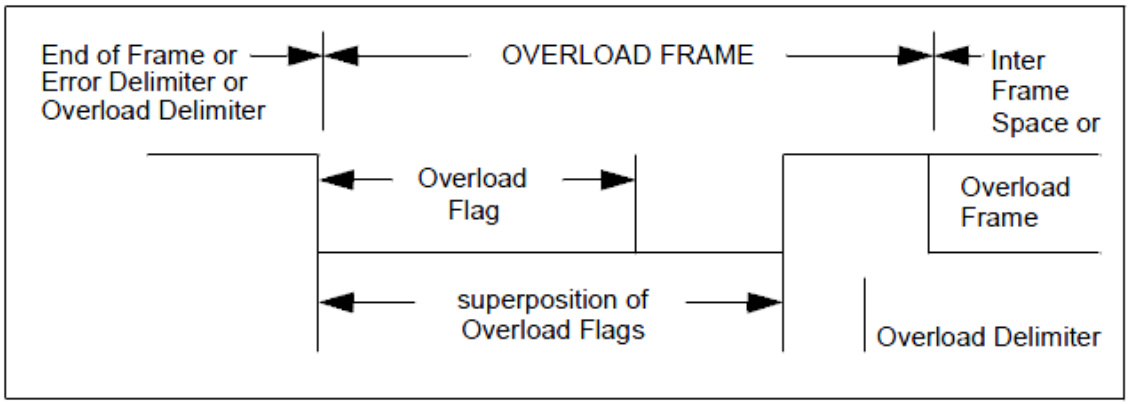


Figura 1.9 – Formato de la trama de sobrecarga [3]

1.4.6 ESPACIO ENTRE TRAMAS

El Interframe Space o espacio entre tramas separa dos tramas sucesivas transmitidas al bus CAN. Consiste en al menos 3 bits a nivel alto, como se muestra en las Figuras 1.10 y 1.11. El espacio entre tramas proporciona tiempo a los nodos para procesar internamente el mensaje recibido previamente, antes del comienzo de la siguiente trama. Si el nodo transmisor está en el estado de error pasivo, se insertan 8 bits a nivel alto adicionales en el espacio entre tramas antes de que el nodo transmita otro mensaje. Este periodo se conoce como campo de transmisión suspendida y proporciona tiempo a otros nodos de la red para tomar el control del bus.

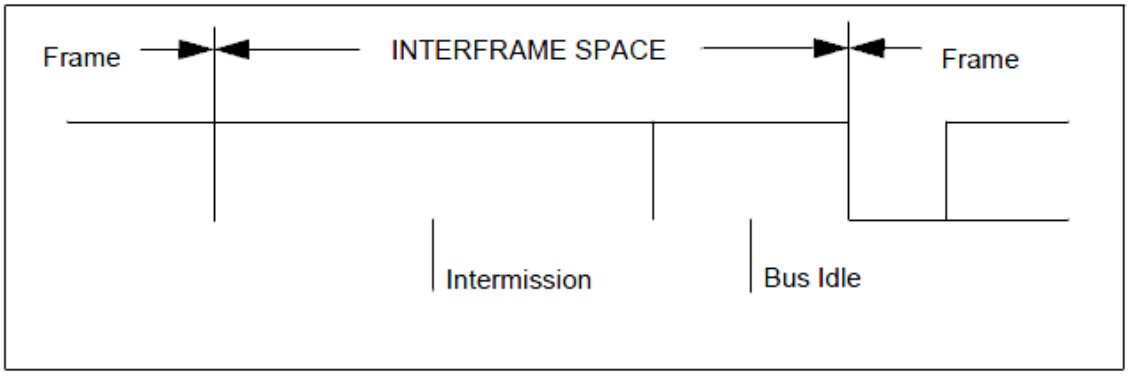


Figura 1.10 – Interframe Space para nodos que no están en estado de error pasivo [3]

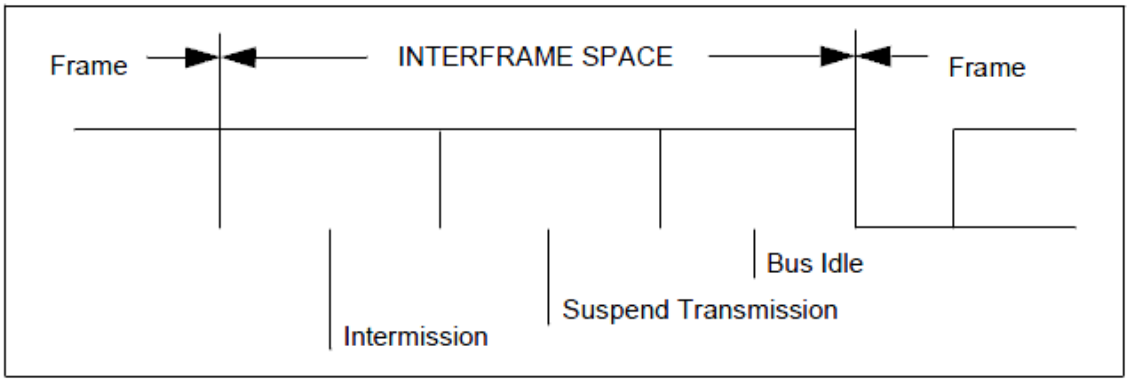


Figura 1.11 – Interframe Space para nodos que están en estado de error pasivo [3]

En este proyecto solo se van a manejar tramas de datos. El módulo ECAN de la familia de microprocesadores PIC24H es compatible con la especificación CAN 2.0B activa y además proporciona capacidades de filtrado de mensajes mejoradas, por lo tanto se podrían transmitir y recibir tramas de datos tanto estándar como extendidas. Sin embargo, ya que la centralita del vehículo trabaja con tramas de datos estándar, este es el único tipo de trama que se ha manejado.

CAPITULO 2: DESARROLLO DEL HARDWARE

2.1 UNIDAD DE CONTROL DE MOTOR; DATOS A MONITORIZAR

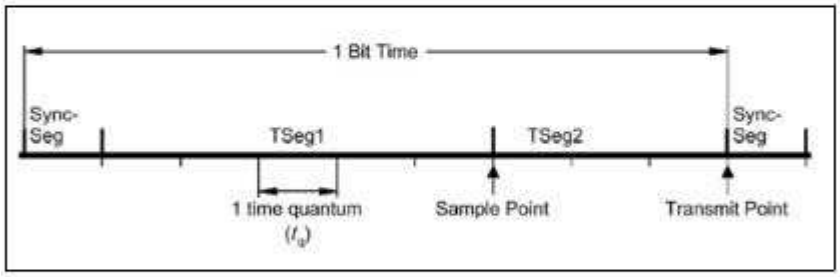
Los datos que se quieren monitorizar son proporcionados por la centralita del vehículo o ECU (Engine Control Unit) a través del bus CAN de comunicaciones. El modelo de ECU es el MS3 Sport de Bosch.

En el manual que proporciona Bosch, se especifica la velocidad a la que los datos son transmitidos por el bus CAN, así como el tiempo de bit. Como puede verse en la Figura 2.1, la velocidad a la que se transmiten los datos es de 1Mbit/s, lo que resulta en un tiempo de bit de 1 microsegundo. El periférico ECAN del microprocesador PIC ha sido programado para cumplir con estos requerimientos, como se explicará más adelante.

Technical specifications / *technische Spezifikation:*

- Frame Type Standard Frame (11 Bit Identifier)
- bus speed 1MBit/s
- processor time slice (Tcpu) 25ns, 40MHz Clock
- Baud-Rate Prescaler (BRP) 1 (tq = 2*(BRP+1)*Tcpu = 100ns)
- Resynchronization Jump Width (SJW) 2 (Tsync = (SJW+1)*tq = 300ns)
- Time Segment before Sample Point (TSEG1) 5 (Tseg1 = (TSEG1+1)*tq = 600ns)
- Time Segment after Sample Point (TSEG2) 2 (Tseg2 = (TSEG2+1)*tq = 300ns)
- Sample Point (Tsync + Tseg1) / (Tsync + Tseg1 + Tseg2) = 700ns / 1000ns = 70%

Sample Point:



Bit timings:

bit time	$= t_{Sync\text{-}Seg} + t_{TSeg1} + t_{TSeg2}$
$t_{Sync\text{-}Seg}$	$= 1 \times t_q$
t_{TSeg1}	$= (TSEG1 + 1) \times t_q$
t_{TSeg2}	$= (TSEG2 + 1) \times t_q$
t_q	$= (BRP + 1) \times 2^{(1\text{-}CPS)} \times t_{XCLK}$

Figura 2.1 – Temporización del bus CAN [4]

Los datos vienen codificados y encapsulados en tramas según indican las siguientes tablas. La aplicación programada en LabVIEW debe realizar las operaciones necesarias según cada trama para obtener el valor real de la variable física que se quiere monitorizar.

Tabla 1 – Tiempo de inyección de cada cilindro del motor [4]

		ID = 0x770	Injection		
byte	row	label	range, conversion formula	type	raster
0	-	ti_1	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
1	-	ti_2	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
2	-	ti_3	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
3	-	ti_4	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
4	-	ti_5	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
5	-	ti_6	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
6	-	ti_7	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
7	-	ti_8	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms

Tabla 2 – Otros parámetros de inyección [4]

		ID = 0x771	Injection		
byte	row	label	range, conversion formula	type	raster
0	-	tibase	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
1	-	tibatt_o	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
2	-	timap	range=0..25.5[ms], phys = int * 25,5 / 255 ms	unsigned	5ms
3	-	injang	range=0..720[°KW], phys = int * 720 / 256 °KW	unsigned	5ms
4	-	injoff	range=0..255, phys = int * 1	unsigned	5ms
5	-	lamctrl_k	range=0..2, phys = int * 2 / 255	unsigned	5ms
6	-	lamctrl_2k	range=0..2, phys = int * 2 / 255	unsigned	5ms
7	-	free			5ms

Tabla 3 – Ángulos de encendido de cada cilindro del motor [4]

		ID = 0x772	Ignition		
byte	row	label	range, conversion formula	type	raster
0		ign_1	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
1		ign_2	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
2		ign_3	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
3		ign_4	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
4		ign_5	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
5		ign_6	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
6		ign_7	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
7		ign_8	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms

Tabla 4 – Datos de ángulo de encendido, revoluciones y aceleración [4]

		ID = 0x773	Ignition / Rev / Ath		
byte	row	label	range, conversion formula	type	raster
0	-	ignbase	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
1	-	ignmap	range=-96..95.25[°KW], phys = 191.25 * int / 255 [°KW]	signed	5ms
2	-	tdwell	range=0..25,5[ms], phys = int * 25.5 / 255 [ms]	unsigned	5ms
3	-	rev.msb	range=0..32767 [rpm], phys = int * 32767.5 / 65535 [kph]	unsigned	5ms
4	-	rev.lsb	range=0..32767 [rpm], phys = int * 32767.5 / 65535 [kph]	unsigned	5ms
5	-	ath	range=0..100[%], phys = int * 100 / 256 [%]	unsigned	5ms
6	-	dath	range=-1536..1524 [%/s], phys = int * 3060 / 255 [%/s]	signed	5ms
7	-	free			5ms

Tabla 5 – Párametros de los sensores Lambda (gases de escape) [4]

		ID = 0x774	Lambda		
byte	row	label	range, conversion formula	type	raster
0	-	lami	range=-32..31.8 [%], phys = int * 64 / 256 [%]	signed	5ms
1	-	lami_2	range=-32..31.8 [%], phys = int * 64 / 256 [%]	signed	5ms
2	-	lamp	range=-32..31.8 [%], phys = int * 64 / 256 [%]	signed	5ms
3	-	lamp_2	range=-32..31.8 [%], phys = int * 64 / 256 [%]	signed	5ms
4	-	lam_f	range=0..2, phys = int * 2 / 255	unsigned	5ms
5	-	lam_2f	range=0..2, phys = int * 2 / 255	unsigned	5ms
6	-	lammap	range=0..2], phys = int * 2 / 255	unsigned	5ms
7	-	free			5ms

Tabla 6 – Velocidad de cada rueda y velocidad del vehículo [4]

		ID = 0x775	Speed		
byte	row	label	range, conversion formula	type	raster
0	-	speed.msb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms
1	-	speed.lsb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms
2	-	speedfl	range=0..512 [kph], phys = int * 512 / 256 [kph]	unsigned	5ms
3	-	speedfr	range=0..512 [kph], phys = int * 512 / 256 [kph]	unsigned	5ms
4	-	speedrl	range=0..512 [kph], phys = int * 512 / 256 [kph]	unsigned	5ms
5	-	speedrr	range=0..512 [kph], phys = int * 512 / 256 [kph]	unsigned	5ms
6	-	free			5ms
7	-	free			5ms

Tabla 7 – Datos de vuelta (distancia, tiempo por vuelta, mejor vuelta) [4]

		ID = 0x776	Lapfunc		
byte	row	label	range, conversion formula	type	raster
0	-	lapdist.msb	range=0..65535[m], phys = int * 1 [m]	unsigned	5ms
1	-	lapdist.lsb	range=0..65535[m], phys = int * 1 [m]	unsigned	5ms
2	-	laptime.msb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms
3	-	laptime.lsb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms
4	-	laptimediff.msb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms
5	-	laptimediff.lsb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms
6	-	laptimefast.msb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms
7	-	laptimefast.lsb	range=0..655,35 [s], phys = int / 100 [s]	unsigned	5ms

Tabla 8 – Marcha y aceleraciones laterales [4]

		ID = 0x777	Gear / Dashboard / Acceleration		
byte	row	label	range, conversion formula	type	raster
0	-	gear	range=0..255, phys = int * 1	unsigned	5ms
1	-	gcstate	range=0..255, phys = int * 1	unsigned	5ms
2	-	gearratio	range=0..16, phys = int * 16 / 256	unsigned	5ms
3	-	gearcut_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	5ms
4	-	ddugear	range=0..255, phys = int * 1 (ASCII value of current gear)	unsigned	5ms
5	-	accx_f	range=-4..3.96 [g], phys = int * 8 / 255 [g]	signed	5ms
6	-	accy_f	range=-4..3.96 [g], phys = int * 8 / 255 [g]	signed	5ms
7	-	accz_f	range=-4..3.96 [g], phys = int * 8 / 255 [g]	signed	5ms

Tabla 9 – Datos de control de tracción [4]

		ID = 0x778	Traction Control		
byte	row	label	range, conversion formula	raster	raster
0	-	tcpfac	range=-100..99.21 [%], phys = int * 200 / 256 [%]	signed	5ms
1	-	tcsw	range=0..255, phys = int * 1	unsigned	5ms
2	-	splisp	range=0..20[%], phys = int * 20 / 255 [%]	unsigned	5ms
3	-	slra	range=0..20[%], phys = int * 20 / 255 [%]	unsigned	5ms
4	-	vrear.msb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms
5	-	vrear.lsb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms
6	-	vref.msb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms
7	-	vref.lsb	range=0..512 [kph], phys = int * 512 / 65536 [kph]	unsigned	5ms

Tabla 10 – Parámetros del acelerador electrónico [4]

		ID = 0x779	Electronic Throttle Control		
byte	row	label	range, conversion formula	type	raster
0	-	etb	range=0..100[%], phys = int / 2 [%]	unsigned	5ms
1	-	etb_sp	range=0..100[%], phys = int / 2 [%]	unsigned	5ms
2	-	aps	range=0..100[%], phys = int / 2 [%]	unsigned	5ms
3	-	-	-	unsigned	5ms
4	-	-	-	unsigned	5ms
5	-	camshaftpos	range=0...128 [°KW], phys = int * 128 / 256 [°KW]	unsigned	5ms
6	-	batt_u	range=0..18.0272 [V], phys = int * 18.0272 / 256 [V]	unsigned	5ms
7	-	lap_c	range=0..255, phys = int * 1	unsigned	5ms

Tabla 11 – Bytes de estado y bits de diagnóstico [4]

		ID = 0x77A	State-Bytes, Diag-Bits		
byte	row	label	range, conversion formula	type	raster
0	-	row counter		unsigned	5ms
1	-	*state byte 1*	injcut_b	bit	5ms
			injcutin_b	bit	5ms
			injenrich_b	bit	5ms
			injstartphase_b	bit	5ms
			lamctrl_b	bit	5ms
			lamctrl_2b	bit	5ms
			gearcut_b	bit	5ms
			tc_b	bit	5ms
2	-	*state byte 2*	idle_b	bit	5ms
			lap_b	bit	5ms
			laptrig_b	bit	5ms
			mil_b	bit	5ms
			oillamp_b	bit	5ms
			phsok_b	bit	5ms
			phsokset_b	bit	5ms
			speedlimit_b	bit	5ms
3	-	*state byte 3*	ignoff_b	bit	5ms
			rev_b	bit	5ms
			revlimit_b	bit	5ms
			startend_b	bit	5ms
			knockadaptenable_b	bit	5ms
			knockenable_b	bit	5ms
			etbsys_e	bit	5ms
			free		5ms
4	1	*diagnosis byte 1*	pinX_b: 4, 5, 6, 7, 8, 9, 18, 19	bit	10ms
5		*diagnosis byte 2*	pinX_b: 20, 21, 22, 23, 24, 25, 28, 30	bit	10ms
6		*diagnosis byte 3*	pinX_b: 32, 37, 38, 39, 40, 41, 42, 44	bit	10ms
7		*diagnosis byte 4*	pinX_b: 45, 46, 47, 48, 54, 55, 56, 57	bit	10ms
4	2	*diagnosis byte 5*	pinX_b: 61, 63, 64, 65, 66, 67, 75, 76	bit	10ms
5		*diagnosis byte 6*	pinX_b: 82, 86, 88, 89, 90, 94, 95, 96	bit	10ms
6		*diagnosis byte 7*	pinX_b: 97, 102, 103, 104, 105, 110, 111, 112	bit	10ms
7		*diagnosis byte 8*	pinX_b: 113, 114, 115, 116, 117, 118, 120, 121	bit	10ms

Tabla 12 – Datos de temperaturas, presiones, consumo de combustible y tensiones de test [4]

		ID = 0x77B	Temperatures, Pressures, Fuel, Diagnosis Voltages		
byte	row	label	range, conversion formula	type	raster
0	-	row counter		unsigned	5ms
1	1	pcrank_f	range=0..1275 [mbar], phys = int * 1275 / 255 [mBar]	unsigned	30ms
2		poil_f	range=0..13,107 [bar], phys = int * 13,107 / 255 [bar]	unsigned	30ms
3		-	-	unsigned	30ms
4		pamb_f.msb	range=0..6553,5 [mBar], phys = int * 6553,5 / 65535 [mBar]	unsigned	30ms
5		pamb_f.lsb	range=0..6553,5 [mBar], phys = int * 6553,5 / 65535 [mBar]	unsigned	30ms
6		pfuel_f	range=0..13,107 [bar], phys = int * 13,107 / 255 [bar]	unsigned	30ms
7		mappos	range=0..255, phys = int * 1	unsigned	30ms
1	2	fuellap.msb	range=0..23,456 [l], phys = int * 23,456 / 65536 [l]	unsigned	30ms
2		fuellap.lsb	range=0..23,456 [l], phys = int * 23,456 / 65536 [l]	unsigned	30ms
3		fueltank.msb	range=-187,648..187,642 [l], phys = int * 375,296/ 65536 [l]	signed	30ms
4		fueltank.lsb	range=-187,648..187,642 [l], phys = int * 375,296/ 65536 [l]	signed	30ms
5		tair	range=-40..215 [°C], phys = int - 40 [°C]	unsigned	30ms
6		tfuel	range=-40..215 [°C], phys = int - 40 [°C]	unsigned	30ms
7		toil	range=-40..215 [°C], phys = int - 40 [°C]	unsigned	30ms
1	3	-	-	unsigned	30ms
2		-	-	unsigned	30ms
3		tmot	range=-40..215 [°C], phys = int - 40 [°C]	unsigned	30ms
4		tex	range=-40..1235 [°C], phys = int * 5 - 40 [°C]	unsigned	30ms
5		tex_2	range=-40..1235 [°C], phys = int * 5 - 40 [°C]	unsigned	30ms
6		ddu leds	oillamp_b: 6, battlow_b:5, shled5_b:4, shled4_b:3, shled3_b:2, shled2_b:1, shled1_b:0	unsigned	30ms
7		pin16_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
1	4	pin17_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
2		pin11_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
3		pin13_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
4		pin26_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
5		pin29_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
6		pin31_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
7		pin34_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
1	5	pin35_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
2		pin49_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
3		pin69_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
4		pin70_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
5		pin74_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
6		pin80_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
7		pin84_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
1	6	pin85_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
2		pin92_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
3		pin93_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
4		pin100_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
5		pin101_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
6		pin109_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms
7		pin121_u	range=0..5 [V], phys = int * 5 / 255 [V]	unsigned	30ms

Cabe destacar que no todos los datos que aquí se reflejan son transmitidos finalmente por el sistema, por varios motivos, bien porque son datos que no tienen interés en los ensayos realizados o bien porque son datos que se refieren a sistemas que no integra el coche en estos momentos. Solamente se transmiten las tramas con identificadores 0x773, 0x775, 0x777, 0x77A y 0x77B.

2.2 DIAGRAMA DE BLOQUES

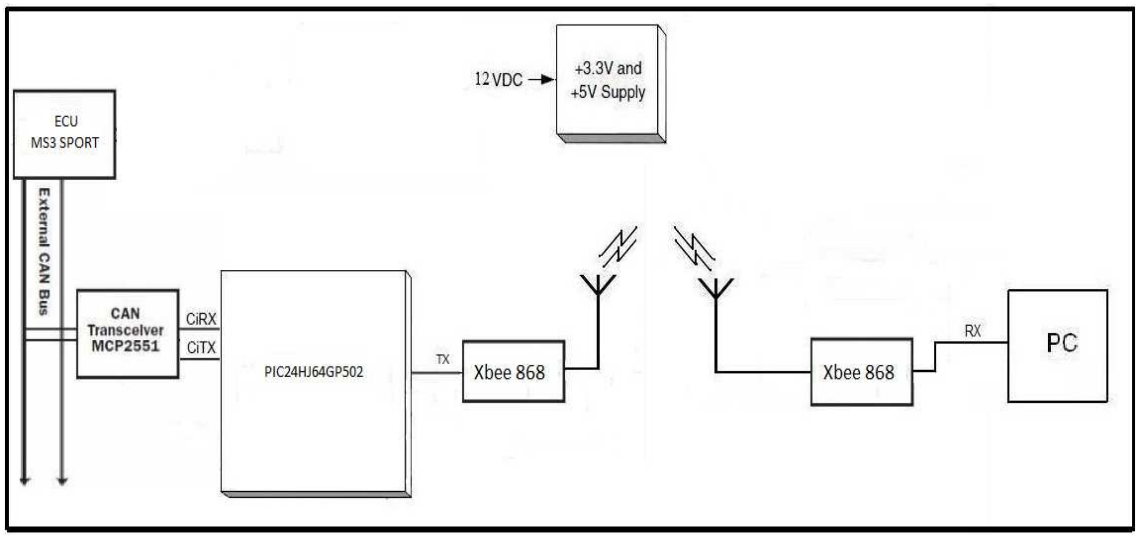


Figura 2.3 – Diagrama de bloques

En la figura 2.3 se representa el diagrama de bloques básico de la aplicación, el cual constituye la parte hardware del sistema y que se compone de tres grandes bloques: el bloque de alimentación, el bloque del microprocesador que es el que se encarga de toda la gestión de los datos y el bloque de radio. En la figura 2.4 se muestra como está montado el sistema en el coche a excepción del módulo receptor de radio y el PC.

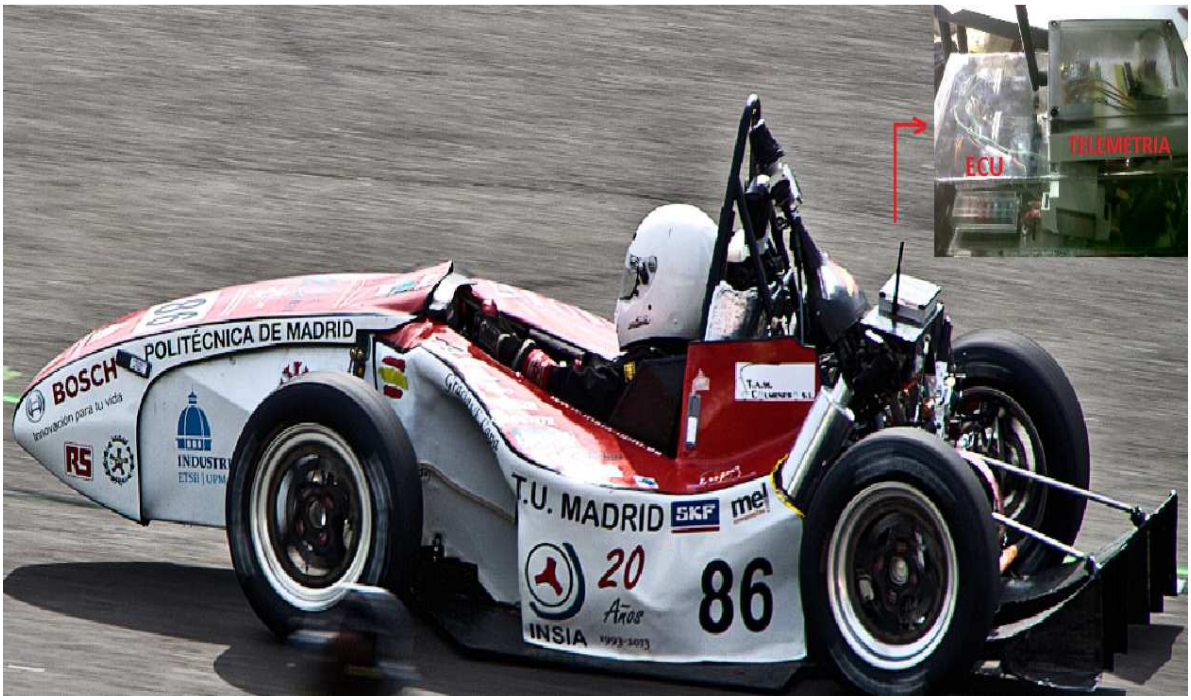


Figura 2.4 – Sistema de telemetría montado sobre el vehículo

A continuación se explica en detalle cada parte del diagrama de bloques.

2.3 MICROPROCESADOR

El microprocesador que se ha elegido para este proyecto es el PIC24HJ64GP502 de Microchip. La función del microprocesador es la de recibir las tramas procedentes del bus CAN de la centralita del vehículo, extraer los datos de cada trama y enviarlos al transmisor de radio por línea serie. El motivo de elegir este microprocesador es fundamentalmente porque integra un periférico específico para la gestión de mensajes CAN, el cual se explicará con más detalle más adelante y también porque es compatible con la tarjeta de desarrollo Microstick, la cual ha facilitado mucho el desarrollo de la parte software del proyecto al permitir la depuración del código sin necesidad de tener implementada toda la parte hardware del proyecto.

La CPU de la familia de microprocesadores PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 y PIC24HJ128GPX02/X04 presentan una arquitectura Harvard modificada de 16 bit con un juego de instrucciones y modos de direccionamiento mejorados. La CPU tiene una palabra de instrucción de 24 bits con un campo de código de operación de longitud variable. El contador de programa tiene una longitud de 23 bits y es capaz de direccionar hasta 4Mx24 bits de memoria de programa. La cantidad de memoria de programa varía según el dispositivo concreto elegido. Aparte de esto, esta familia de microprocesadores presenta los siguientes recursos:

Manejo del reloj:

- Dos osciladores internos.

-PLL programable.

- Fail-safe clock monitor (FSCM).

- Watchdog timer independiente.

- Modos de bajo consumo.

Características Analógicas Avanzadas:

- ADC de 10/12 bits con tasas de conversión desde 500ksps hasta 1.1 Msps:
  - Hasta trece canales de entrada al ADC y cuatros Sample&Hold.
  - Fuentes de disparo independientes.
- Comparadores de 150 ns:
  - Dos módulos comparadores analógicos.
  - DAC de 4 bit para los comparadores analógicos.

Entrada/Salida:

- Opción de remapear pines por software.

- Pines tolerantes a 5V.

- Múltiples interrupciones externas.

Entrada/Salida digital:

- Hasta 85 pines digitales de entrada/salida programables.

- Los pines de salida pueden soportar desde 3V hasta 3.6V.

- Todos los pines que son entradas digitales toleran 5V.

Periféricos:

- Modulo de comprobación de redundancia cíclica (CRC).

- 5 timers de 16 bits y hasta 2 timers de 32 bits.

- Hasta cuatro módulos de captura.

- Hasta cuatro módulos de comparación.

- Módulo de reloj en tiempo real y calendario (RTCC).

Módulos de Comunicación:

- Puerto maestro paralelo (PMP).

- Dos módulos UART (10Mbps):

    - Soporta protocolo LIN 2.0.
    - Soporte de RS-232, RS-485 e IrDA.

- Dos módulos SPI de cuatro hilos (15Mbps).

- Módulo CAN mejorado (ECAN) (1Mbaud) con soporte de CAN 2.0B.

- Módulo I2C (100K, 400K y 1Mbaud) con soporte SMBus.

Acceso Directo a Memoria:

- 8 canales hardware DMA.

- UART, SPI, ADC, ECAN, IC, OC, INT0.

Soporte de Modo Depurador:

- Programación en circuito y en aplicación.

- Dos puntos de ruptura en el programa.

2.3.1 REGLAS DE DISEÑO BASICAS PARA LOS CONTROLADORES DIGITALES DE SEÑAL DE 16-BITS

Para empezar a trabajar con la familia de controladores digitales de señal (DSCs) de 16 bits PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 y PIC24HJ128GPX02/X040, hay que prestar atención a la conexión de una serie de pines, antes de proceder al desarrollo de aplicaciones. A continuación se muestra una lista de los pines que se deben conectar siempre:

Todos los pines de VDD y VSS:

Además de conectar todos los pines de este tipo a su respectivo nivel de tensión, es necesario colocar entre cada par de pines de VDD y VSS un condensador de desacoplo de una valor recomendado de 100nF y que aguante hasta 20V. Estos condensadores se deben situar lo más cerca posible de los pines en la placa de circuito impreso.

Todos los pines de AVDD y AVSS (aunque no se use el módulo ADC):

También entre cada par de pines de este tipo hay que colocar condensadores de desacoplo con las mismas características que los anteriores.

Pin MCLR:

Este pin sirve para dos funciones específicas del microprocesador. Por un lado sirve como reset del dispositivo y por otro para la programación y depuración del mismo.

Pines PGECx/PGDEx:

Estos pines se usan para la programación y depuración del microprocesador. La manera en la que deben conectarse se explica más adelante, en la sección 2.3 de este documento.

En la figura 2.5 se muestran las conexiones mínimas recomendadas que debe tener el microprocesador:

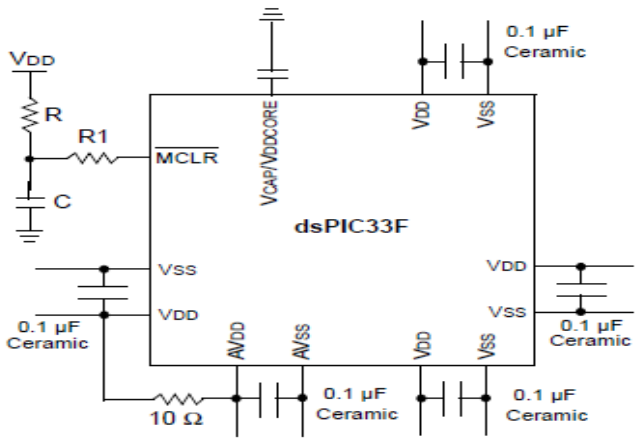


Figura 2.5- Conexiones mínimas recomendadas del microprocesador [5]

2.3.2 ORGANIZACIÓN DE LA MEMORIA

Como corresponde a los dispositivos con arquitectura Harvard, la familia de microprocesadores PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 y PIC24HJ128GPX02/X040 presenta varios conjuntos de buses diferentes para acceso a memoria, habitualmente dos:

- Uno para acceso a memoria de programa.

- Otro para acceso a memoria de datos.

El espacio de memoria de programa de los dispositivos de la familia PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 y PIC24HJ128GPX02/X040 es de 4 Mbytes y está situado en la parte baja del mapa de memoria (000000h hasta 7FFFFFh), como puede verse en la figura 2.6.

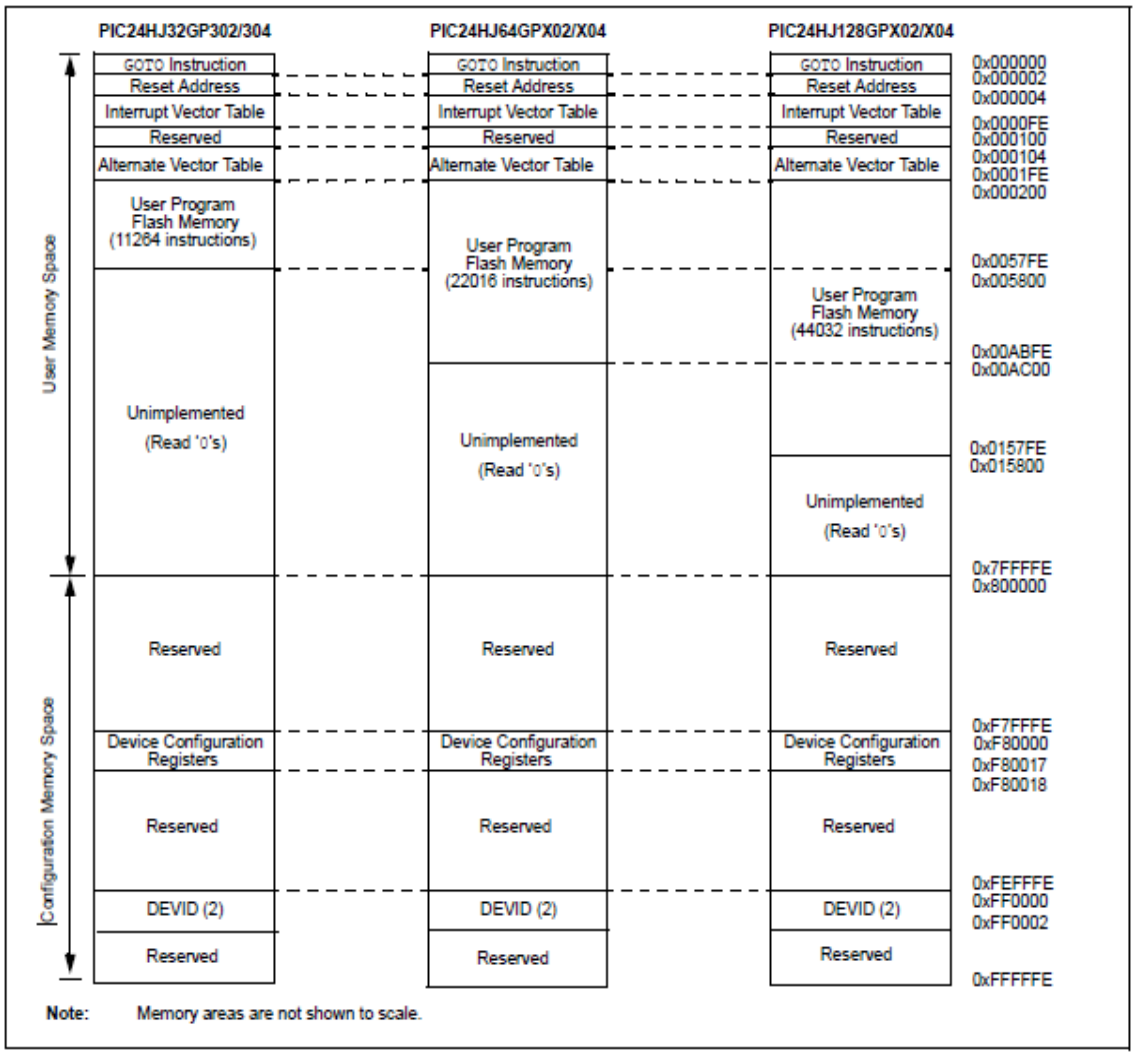


Figura 2.6- Mapa de memoria de programa [5]

En cuanto a la memoria de datos, este microprocesador tiene un espacio de direccionamiento de 16 bits, lo que proporciona un espacio de datos de 64 Kbytes. En la figura 2.7 se muestra el mapa de memoria de datos.

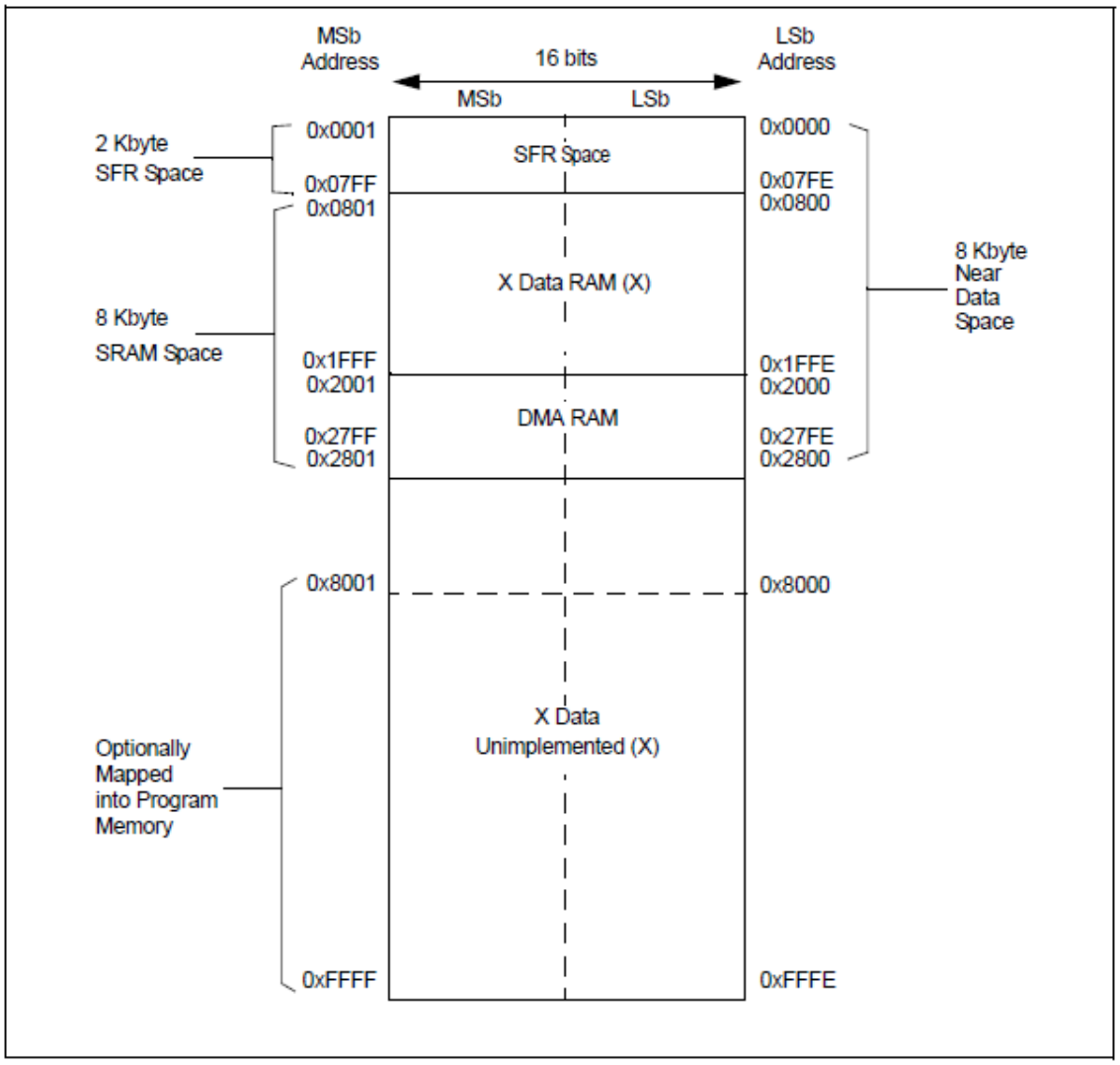


Figura 2.7- Mapa de memoria de datos [5]

2.4 BLOQUE DE ALIMENTACIÓN

El bloque de alimentación tiene como misión proporcionar la tensión de alimentación a los diferentes circuitos integrados de las placas, para lo cual emplea varios reguladores lineales de tensión. Para saber cuántos reguladores lineales eran necesarios se estudiaron las hojas de características de los distintos integrados, determinando que el microprocesador se debe alimentar a 3.3V, el transceiver del bus CAN (MCP2551) se alimenta a 5V y el interfaz de radio soporta tensiones de alimentación entre 3V y 3.6V. Se deben usar por tanto, dos reguladores de tensión, uno de 5 V y otro de 3,3 V.

Para obtener la tensión de alimentación de 5V se ha empleado el modelo de regulador LM1117, en formato SMD.

Para obtener la tensión de alimentación de 3,3V y debido al gran consumo de corriente que puede presentar el módulo de radio (hasta 0,8A trabajando a máxima potencia) se ha elegido el regulador LM350T. Es un regulador ajustable que puede proporcionar una tensión de salida desde 1,2V hasta 33V y una corriente de hasta 3A. Se ha elegido la configuración de resistencias adecuada para obtener una tensión de salida de 3,3V. En la figura 2.8 se muestra el esquema que se ha montado en la placa de circuito impreso.

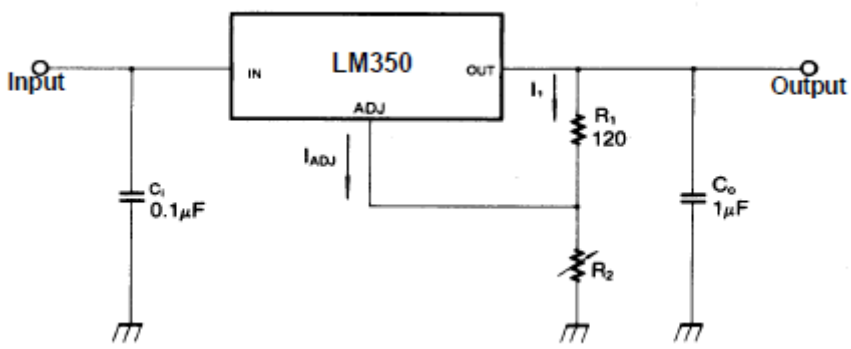


Figura 2.8- Esquemático del regulador de tensión de 3,3V [6]

La tensión de salida se calcula mediante la expresión $V_{OUT} = 1,25V(1+\frac{R2}{R1}) + I_{ADJ} \cdot R2$, siendo la corriente I_{ADJ} menor de $100\mu A$, por lo que el segundo término de la expresión se desprecia a la hora de calcular la tensión de salida.

Como se desea una tensión de salida de 3,3V, se obtiene una relación $\frac{R2}{R1} = 1,64$. Fijando $R2 = 120\Omega$ se necesita una resistencia $R1 = 196,8\Omega$, para lo que se ha utilizado un potenciómetro de $1k\Omega$.

2.5 CONEXIÓN ENTRE EL MICROPROCESADOR Y EL INTERFACE DE RADIO

Tras realizar un estudio del microprocesador y del módulo de interfaz de radio, se llegó a la conclusión de que los niveles de tensión de salida que proporciona el microprocesador son compatibles con los niveles que admite el interfaz de radio. Observando las tablas número 13 y número 14, se comprueba que los niveles de tensión requeridos por el microprocesador y por el interfaz de radio son compatibles. Primero se miran las tensiones a nivel bajo: el microprocesador proporciona a su salida un nivel de tensión máxima a nivel bajo de 0.4V (VOL), y el interfaz de radio soporta como máximo una tensión de VIL = 0,2*VCC = 0,66 V a nivel bajo. En cuanto a las tensiones a nivel alto, el microprocesador proporciona a su salida, como mínimo, una tensión de entre 1,5 y 3V (VOH), dependiendo del consumo de corriente demandado, mientras que el interfaz de radio necesita una tensión mínima de entrada a nivel alto de VIH = 0,8*VCC = 2,64V. Por tanto se demuestra que los dos circuitos son directamente compatibles, es decir, no es necesario colocar ningún circuito adaptador entre ambos.

Tabla 13 - Características eléctricas del interfaz de radio [7]

Symbol	Parameter	Condition	Min	Typical	Max	Units
V_{IL}	Input Low Voltage	All Digital Inputs	-	-	0.2 * VCC	V
V_{IH}	Input High Voltage	All Digital Inputs	0.8 * VCC	-	-	V
V_{OL}	Output Low Voltage	I_{OL} = 2 mA, VCC >= 3.0 V	-	-	0.18*VCC	V
V_{OH}	Output High Voltage	I_{OH} = 2 mA, VCC >= 3.0 V	0.82*VCC	-	-	V
I_{IN}	Input Leakage Current	V_{IN} = VCC or GND, all inputs, per pin	-	-	0.5uA	uA
Iout	Output Current	Dout, DIO(0,1,2,3,6,7,8), On/Sleep	-	-	8	mA
Iout	Output Current	DIO9, DIO10, DTR	-	-	16	mA
Iout	Output Current	DIO4, DIO5	-	-	2	mA
I-TX	Max Tx current Draw over voltage and Temp	VCC = 3.3v Power = +25dBm	-	500	800	mA

Tabla 14 - Características eléctricas del microprocesador [5]

DC CHARACTERISTICS			Standard Operating Conditions: 3.0V to 3.6V (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for Industrial -40°C ≤ TA ≤ +125°C for Extended				
Param.	Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
DO10	VOL	**Output Low Voltage** I/O Pins: 2x Sink Driver Pins - RA2, RA7-RA10, RB10, RB11, RB7, RB4, RC3-RC9	—	—	0.4	V	IOL ≤3 mA, VDD = 3.3V See **Note 1**
		Output Low Voltage I/O Pins: 4x Sink Driver Pins - RA0, RA1, RB0-RB3, RB5, RB6, RB8, RB9, RB12-RB15, RC0-RC2	—	—	0.4	V	IOL ≤6 mA, VDD = 3.3V See **Note 1**
		Output Low Voltage I/O Pins: 8x Sink Driver Pins - RA3, RA4	—	—	0.4	V	IOL ≤10 mA, VDD = 3.3V See **Note 1**
DO20	VOH	**Output High Voltage** I/O Pins: 2x Source Driver Pins - RA2, RA7-RA10, RB4, RB7, RB10, RB11, RC3-RC9	2.4	—	—	V	IOH ≥ -3 mA, VDD = 3.3V See **Note 1**
		Output High Voltage I/O Pins: 4x Source Driver Pins - RA0, RA1, RB0-RB3, RB5, RB6, RB8, RB9, RB12-RB15, RC0-RC2	2.4	—	—	V	IOH ≥ -6 mA, VDD = 3.3V See **Note 1**
		Output High Voltage I/O Pins: 8x Source Driver Pins - RA4, RA3	2.4	—	—	V	IOH ≥ -10 mA, VDD = 3.3V See **Note 1**
		Output High Voltage I/O Pins: 2x Source Driver Pins - RA2, RA7-RA10, RB4, RB7, RB10, RB11, RC3-RC9	1.5	—	—	V	IOH ≥ -6 mA, VDD = 3.3V See **Note 1**
			2.0	—	—		IOH ≥ -5 mA, VDD = 3.3V See **Note 1**
			3.0	—	—		IOH ≥ -2 mA, VDD = 3.3V See **Note 1**

2.6 CONEXIÓN DEL BUS CAN AL MICROPROCESADOR

2.6.1 TRANSCEIVER MCP2551 DE MICROCHIP

El **MCP2551** es un dispositivo de alta velocidad que sirve como interfaz entre el bus físico y los controladores en un sistema con protocolo CAN. El dispositivo proporciona la capacidad de transmitir y recibir entre muchos nodos conectados a la misma red.

El **MCP2551** cumple con los requerimientos de la norma ISO11898, una especificación estándar que describe las características eléctricas de conexión a una red con nodos, muy usada en automoción para la comunicación CAN a velocidades de hasta 1Mbps.

La mayoría de las aplicaciones de automoción e industriales requieren que los nodos CAN cumplan la especificación ISO11898. Este dispositivo junto con otros componentes CAN, como circuitos de acondicionamiento de señal y alimentación analógica de Microchip, permite al ingeniero realizar un nodo CAN completo usando productos exclusivamente de Microchip.

El **MCP2551** soporta los transitorios de tensión más exigentes de la industria, según la ISO7637, desde -250 hasta +250 voltios, sin circuitos de protección. También es capaz de soportar tensiones desde -40 hasta +40 voltios.

En muchas aplicaciones se utilizan redes CAN con velocidades más bajas (típicamente de 125Kbps o menos), por lo que se pueden implementar sus redes sin necesidad de cables de par trenzado apantallado, reduciendo los costes de la aplicación. El dispositivo permite el control de la rampa de las señales de salida diferencial (CANH y CANL) a través de una resistencia externa, reduciendo la rampa de las señales, se reducen las emisiones de radiofrecuencia y se pueden mantener los niveles de emisión por debajo de los máximos permitidos, incluso sobre par trenzado sin apantallar.

Las señales CANH y CANL están protegidas contra cortocircuitos en la batería y transitorios eléctricos que pueden ocurrir en el bus CAN. Esta característica, previene la destrucción de la etapa de salida del transmisor durante tal condición de fallo. El dispositivo está protegido contra corriente excesiva en la carga mediante circuitos de protección térmica, que deshabilitan a los drivers de salida cuando la temperatura de unión excede de 165 ºC, permaneciendo las demás partes del dispositivo operativas. Otras características incluyen la detección permanente del estado dominante, evitando el bloqueo del bus por un nodo. El **MCP2551** está disponible en encapsulados de 8 pines PDIP y SOIC.

A continuación se muestra un ejemplo de conexión entre el bus CAN y el microprocesador a través del **MCP2551** (figura 2.9):

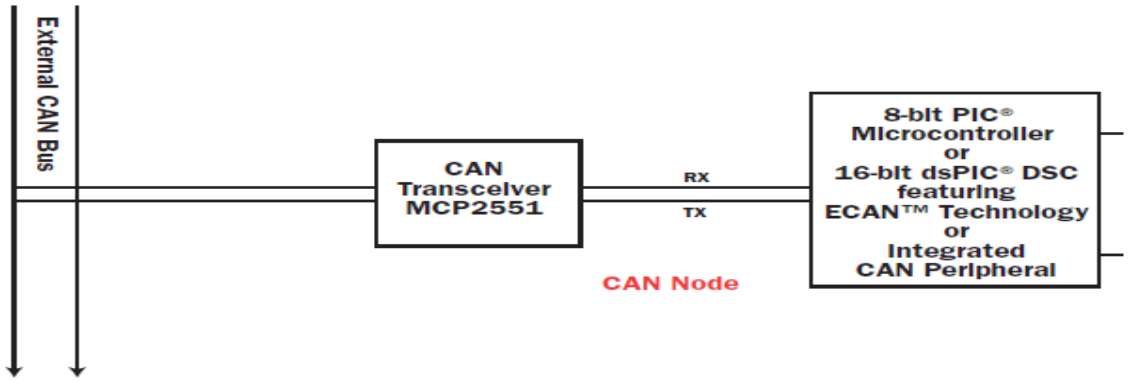


Figura 2.9- Conexión entre el microprocesador y el bus CAN

2.7 TRANSMISOR Y RECEPTOR DE RADIOFRECUENCIA: KIT DE DESARROLLO XBEE PRO 868

Como interfaz de radio se han utilizado los módulos de radio Xbee Pro 868 de la marca Digi, que trabajan a 868 MHz. Para poder hacer pruebas con ellos y desarrollar esta parte del proyecto se ha hecho uso del kit de desarrollo Xbee Pro 868, el cual se compone de dos módulos de radio, uno con conector para antena RPSMA y otro con antena de látigo, una antena RPSMA, dos placas de desarrollo que permiten configurar los chips y realizar pruebas de comunicación y de alcance, dos cables USB que sirven tanto para comunicarse con el chip como para alimentar las placas de desarrollo y un transformador por si se quiere alimentar la placa sin necesidad del USB (normalmente se alimenta el transmisor con el transformador y el receptor se conecta al ordenador mediante USB).

2.7.1 CARACTERÍSTICAS PRINCIPALES

Como características principales de estos módulos se pueden citar las siguientes:

- Rendimiento/Coste:
  - Alcance en entornos urbanos de hasta 550 metros.
  - Alcance en espacio abierto y visión directa de hasta 40km.
  - Potencia de salida de transmisión entre 1mW y 315mW (0 dBm hasta +25 dBm).
  - Sensibilidad del receptor: -112 dBm.
  - Tasa de datos RF: 24kbps.
- Seguridad e interconexión avanzadas:
  - Reintentos y reconocmientos.
  - Soporta topologías punto a punto, punto a multipunto y peer to peer.
  - Encriptación AES de 128 bits.
  - Identificador de red de 16 bits.
- Baja potencia:
  - Corriente de transmisión: 85 a 500mA, dependiendo del nivel de potencia de transmisión seleccionado.
  - Coriente de recepción: 65 mA (@3.3V)
- Fácil uso:
  - Ninguna configuración necesaria para comunicaciones RF.
  - Modos de configuración AT y API Command para la configuración de los parámetros del módulo.
  - Amplio grupo de comandos.
  - Software de configuración y test gratuito (X-CTU).

En la figura 2.10 se presenta el diagrama de bloques del módulo de radio, compuesto básicamente por un procesador y una etapa amplificadora, seguida de un interruptor que controla si el módulo funciona como transmisor o como receptor.

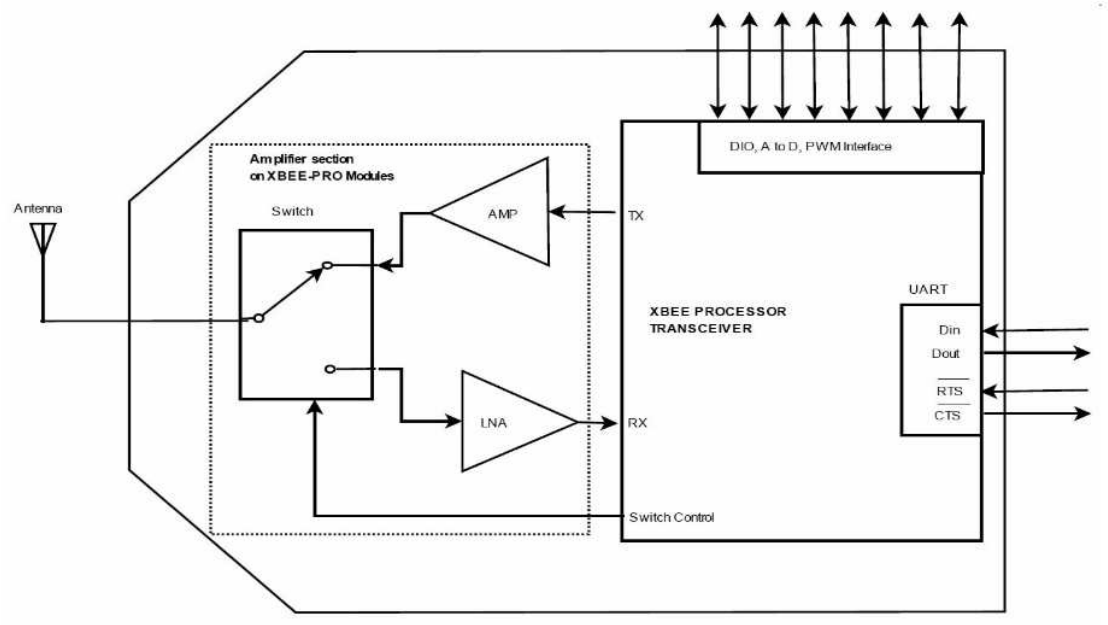


Figura 2.10 – Diagrama de bloques del módulo de radio [7]

2.7.2 FUNCIONAMIENTO

El módulo Xbee proporciona una interfaz serie para un enlace de radiofrecuencia. Puede convertir datos serie en datos RF que pueden ser enviados a cualquier dispositivo en una red RF.

El módulo Xbee se interconecta al dispositivo receptor a través de un puerto serie asíncrono. A través de este puerto serie el módulo puede comunicarse con cualquier dispositivo compatible a nivel de tensión y a nivel lógico o a través de un conversor de niveles con cualquier dispositivo serie.

Cualquier dispositivo que posea una UART puede conectarse directamente al módulo de radio como se indica en la figura 2.11.

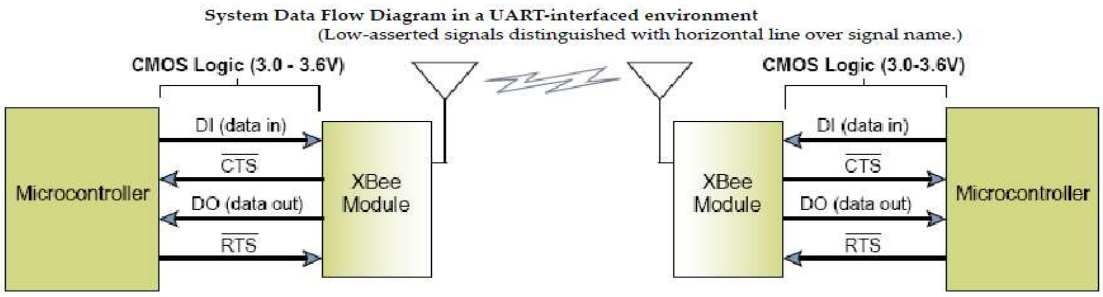


Figura 2.11 – Enlace de radio entre los módulos Xbee [7]

Los datos entran a la UART a través del pin DIN del módulo Xbee mediante una señal serie asíncrona. La señal permanece a nivel alto cuando no se están transmitiendo datos.

La figura 2.12 muestra un ejemplo de la señal que transmite el módulo de radio, aunque el número de bits de parada y el número de bytes de datos es configurable. En la señal de la figura el byte de datos está compuesto de 1 bit de arranque (a nivel bajo), 8 bits de datos (bit menos significativo primero) y 1 bit de parada (a nivel alto). A continuación se muestra un ejemplo de la señal que se transmite a través del chip de radio.

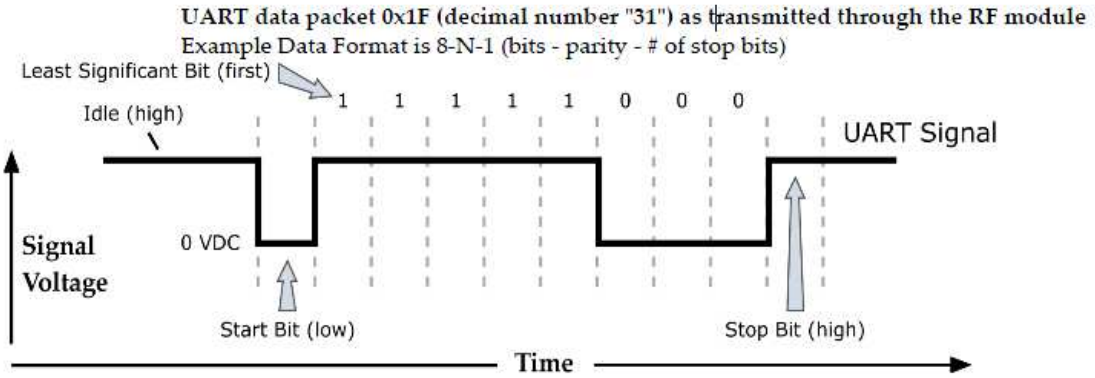


Figura 2.12 – Señal de la UART del módulo del radio [7]

La UART del módulo Xbee lleva a cabo tareas como la comprobación de tiempos y paridad que son necesarias para la comunicación. Las UART de ambos módulos deben ser configuradas con los mismos parámetros (velocidad, paridad, bits de arranque, bits de parada y bits de datos).

Los módulos Xbee poseen buffers para almacenar tanto los datos serie como los RF. El buffer serie almacena los caracteres serie entrantes y los mantiene hasta que pueden ser procesados.

El buffer de transmisión serie almacena los datos que son recibidos a través del enlace RF que serán transmitidos fuera del chip de radio a través de la UART. En la figura 2.13 se muestra un diagrama de bloques de los buffer de transmisión y recepción serie.

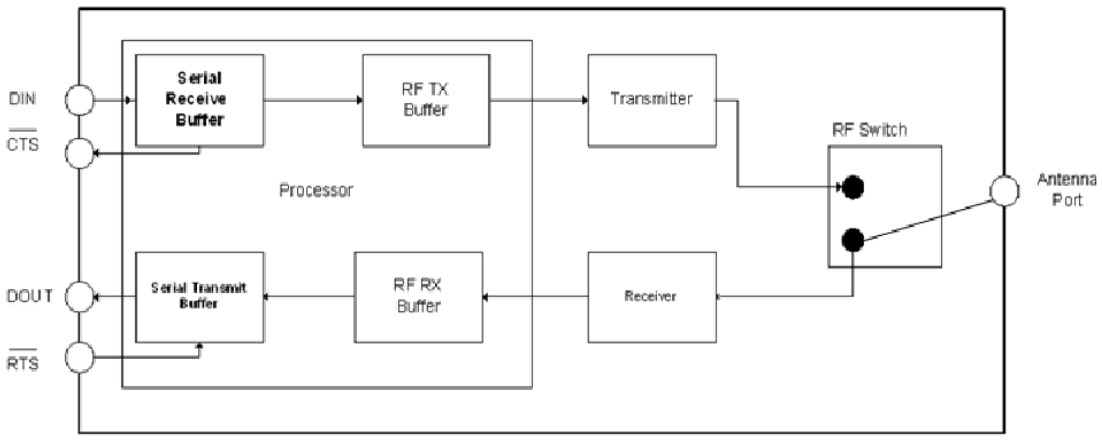


Figura 2.13 – Buffers del módulo de radio [7]

Buffer de recepción serie:

Cuando los datos serie entran al módulo de radio a través del pin DIN, son almacenados en el buffer de recepción serie y permanecen ahí hasta que pueden ser procesados. Bajo algunas circunstancias, el módulo puede no ser capaz de procesar los datos inmediatamente. Si se mandan grandes cantidades de datos al módulo, puede ser necesario implementar un control de flujo mediante el pin CTS para evitar desbordar el buffer.

El buffer puede llenarse y desbordarse si el módulo está recibiendo un flujo constante de datos RF, ya que los datos no saldrán del buffer de recepción serie hasta que el módulo deje de recibir datos RF. Después de transmitir los datos, el módulo puede necesitar retransmitir los datos si no se ha recibido confirmación o si la trama era una retransmisión. Estos problemas pueden retardar el procesamiento de los datos en el buffer de recepción serie.

Buffer de transmisión serie:

Cuando llegan datos RF al chip, estos son almacenados en el buffer de transmisión serie y posteriormente enviados fuera del chip a través del puerto serie. Si el buffer de transmisión serie llega a estar suficientemente lleno como para que no quepa otro paquete de datos, dicho paquete es descartado y por tanto se pierde.

Esto puede ocurrir si la tasa de recepción de datos del interfaz al que está conectado el chip de radio es menor que la tasa de recepción de datos RF por parte del chip, de modo que el módulo de radio recibiría datos más rápido de lo que sería capaz de enviarlos por el puerto serie.

El otro caso en el que se podrían perder datos es si el módulo receptor no permite al chip de radio mandar datos por el puerto serie debido a estar inhabilitado por la señal de control de flujo $\overline{RTS}$.

Para la aplicación que se ha desarrollado no se ha incluido control de flujo ya que se va a controlar mediante una configuración adecuada de la velocidad de envío y recepción de datos que los buffer nunca se llenan. Se va a trabajar por tanto en un modo que tiene el chip de radio llamado modo de operación transparente.

Modo de operación transparente

Cuando el chip trabaja en el modo de operación transparente, actúa como si fuera una línea serie, de manera que todos los datos que son recibidos por la entrada DIN son encolados para ser transmitidos vía radio. Así mismo, cuando los datos RF son recibidos, éstos son inmediatamente transmitidos por el pin DOUT.

Como ya se ha dicho previamente, en la aplicación que se ha desarrollado se va a controlar que nunca se llenen los buffer de los módulos de radio. Al ser una aplicación

de telemetría en un principio se pensó que se deberían visualizar los datos en tiempo real, pero al comprobar que el bus CAN de la centralita manda datos a 1Mbps y que la tasa de datos RF máxima de los módulos de radio es de 24kbps, quedó claro que se perderían datos obligatoriamente. Por lo tanto se decidió configurar todas las comunicaciones serie a 19200bps, con 8 bits de datos, 1 bit de arranque y 1 bit de parada y sin paridad, ya que con esa velocidad se asegura que no se saturan los buffers de los módulos de radio. El siguiente valor de velocidad al que se podría configurar el chip de radio es de 38400bps, que es mayor de 24kbps, por lo que se saturarían los buffers del chip de radio.

El perder datos no es crítico a la hora de monitorizar ciertas variables como presiones y temperaturas o incluso la tensión de la batería del coche, ya que estas variables no van a presentar cambios excesivamente rápidos y por tanto con la velocidad configurada de 19200bps se consiguen monitorizar estas variables sin perder información importante.

Sin embargo, para otro tipo de variables como pueden ser revoluciones del motor, posición del acelerador o velocidades, si es un problema el perder datos.

Por lo tanto se puede concluir que el sistema desarrollado es útil para controlar ciertos parámetros críticos del funcionamiento del motor y dar órdenes al piloto en función de ello, pero que para poder realizar un estudio más profundo y exacto del comportamiento del coche en carrera es necesario un sistema de adquisición de datos que sea capaz de grabar los datos en tiempo real.

Esta función es realizada por un datalogger de la marca AIM, en concreto el modelo EVO4, que va conectado por cable al bus CAN de la centralita y si es capaz de adquirir datos a 1Mbp. Este sistema es ajeno al proyecto desarrollado.

2.8 DISEÑO DE LAS PLACAS DE CIRCUITO IMPRESO

Para el diseño de las placas de circuito impreso se ha utilizado un software libre recientemente creado por la empresa RS llamado DesignSpark PCB. Los motivos para elegir este software han sido, primero el aprender algo nuevo ya que ya se sabía manejar Orcad, segundo que este software permite generar una vista 3D de la PCB que se va a fabricar evitando así que pueda existir cualquier tipo de interferencia entre componentes a la hora de montar la PCB y tercero que es un software libre y gratuito.

El primer paso del proceso es realizar la captura del esquemático del circuito. A continuación se explica detalladamente el proceso que se ha seguido para obtener el esquemático del circuito.

Para ello, una vez abierto el programa se debe pinchar en el menú File→New y en la ventana que aparece seleccionar la opción Schematic Design, como se muestra en la figura 2.14.

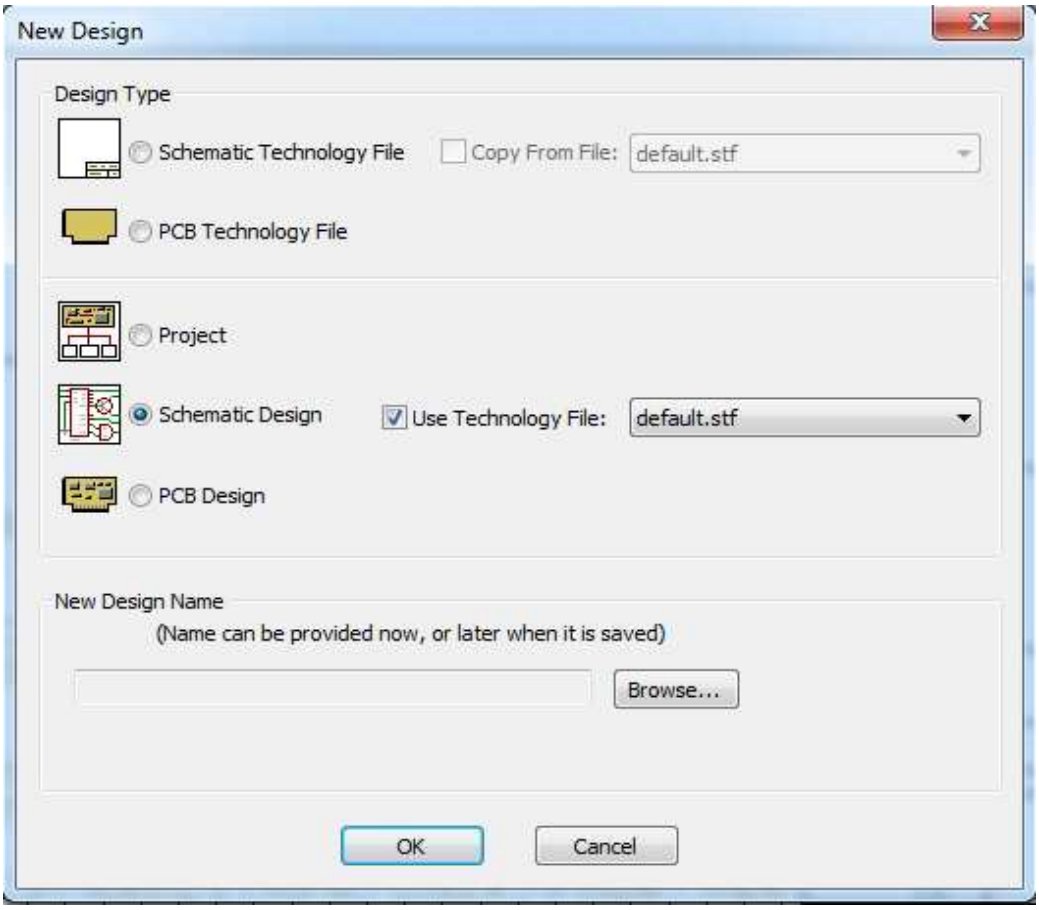


Figura 2.14 – Creación de un nuevo esquemático con DesignSpark

A continuación se van añadiendo los componentes e interconectándolos para crear el esquemático. Como algunos de los componentes que se iban a emplear en el diseño no se encontraban en las librerías que vienen por defecto con el programa, hubo que crear dichos componentes.

Para crear un nuevo componente es necesario crear tres librerías: una con el símbolo del componente para el esquemático (.ssl), otra con la huella del componente para el layout (.psl) y otra que contiene a las dos anteriores y es la que permite utilizar el componente en el diseño (.cml).

Para crear una nueva librería pinchar en el icono Libraries de la barra de herramientas y en la ventana que se despliega como se puede ver en la figura 2.15, elegir el tipo de librería que se quiere crear y pinchar en New Library y elegir el directorio donde se guardará la nueva librería.

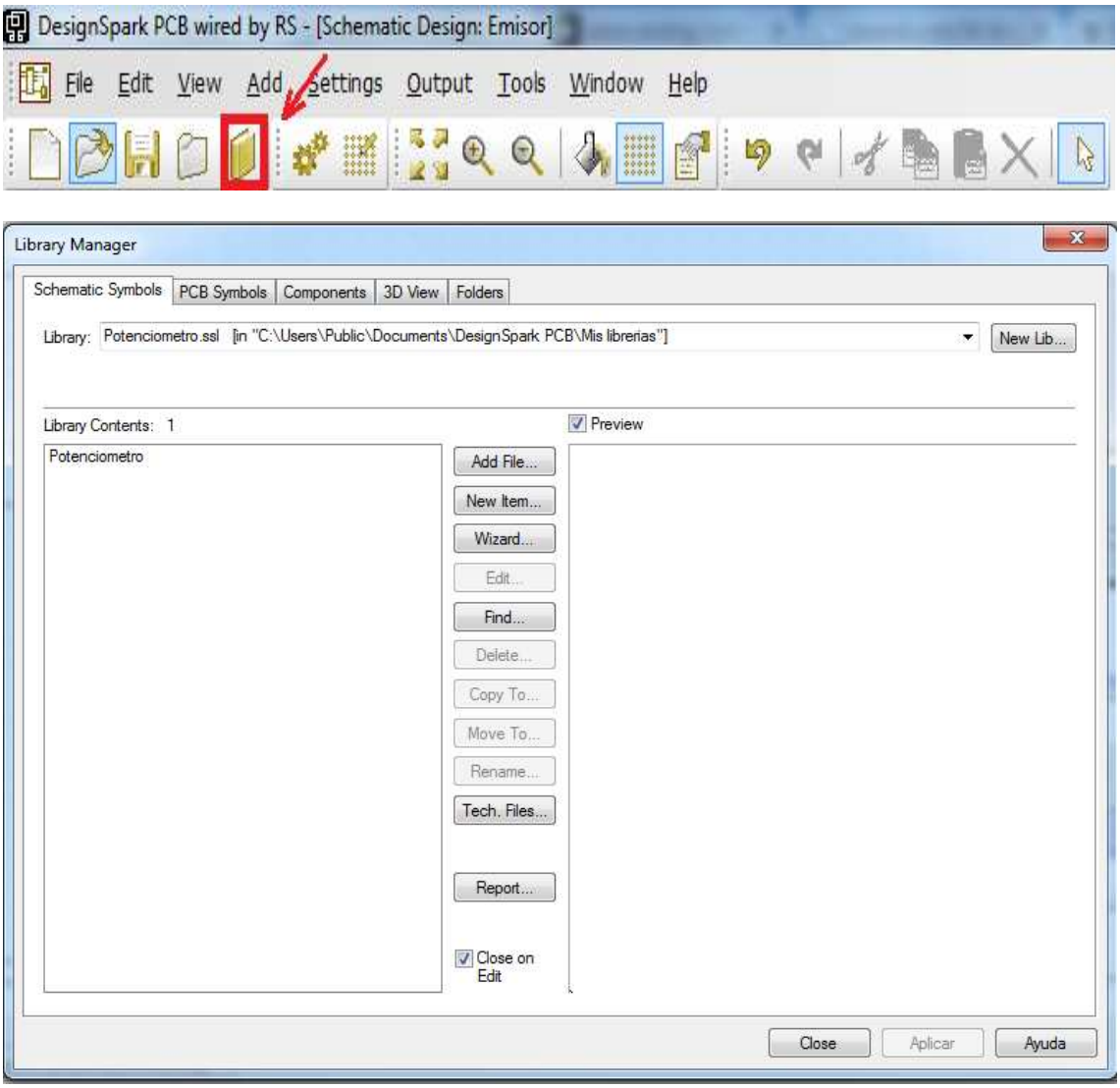


Figura 2.15 – Creación de una nueva librería con DesginSpark

Este proceso se debe repetir para cada una de las tres librerías que se deben crear para obtener el nuevo componente.

A continuación se muestran las imágenes de los componentes que ha sido necesario crear para este diseño en las figuras 2.16, 2.17, 2.18, 2.19, 2.20 y 2.21:

Conector de 2 pines

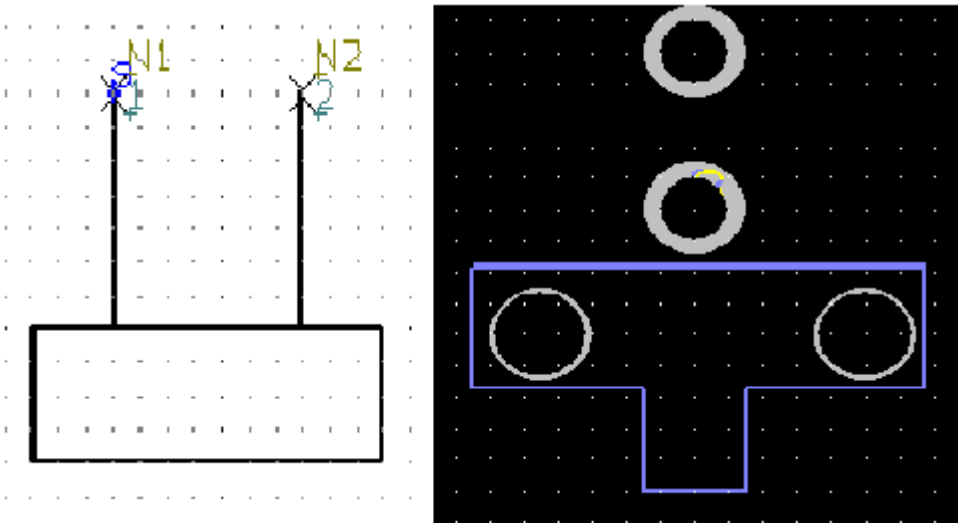


Figura 2.16 – Símbolo y footprint del conector de dos pines

Clema 7 pines

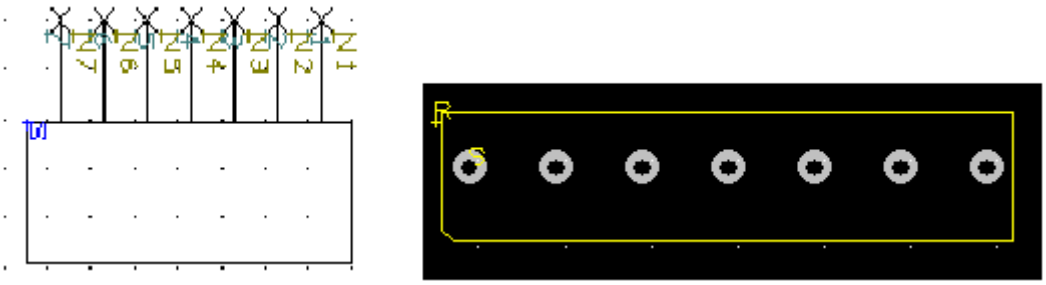


Figura 2.17 – Símbolo y footprint del conector de la clema de siete pines

Condensador electrolítico SMD

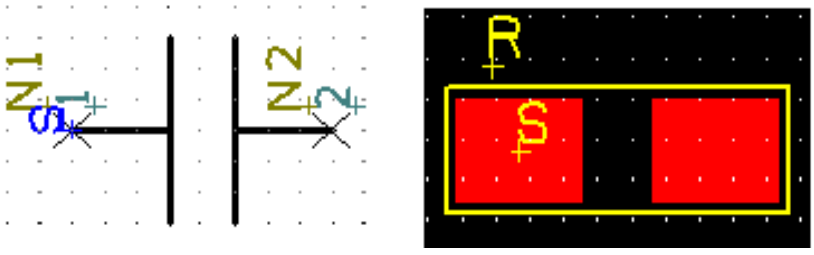


Figura 2.18 – Símbolo y footprint del condensador electrolítico

Pulsador

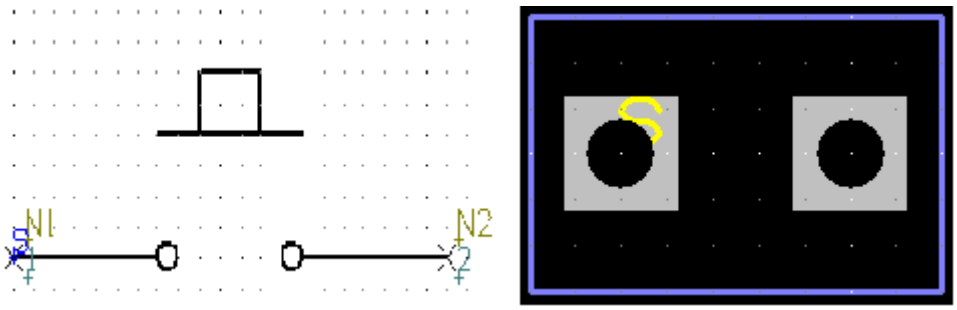


Figura 2.19 – Símbolo y footprint del pulsador

Jumper 3 pines

Figura 2.20 – Símbolo y footprint del jumper de tres pines

Módulo de radio

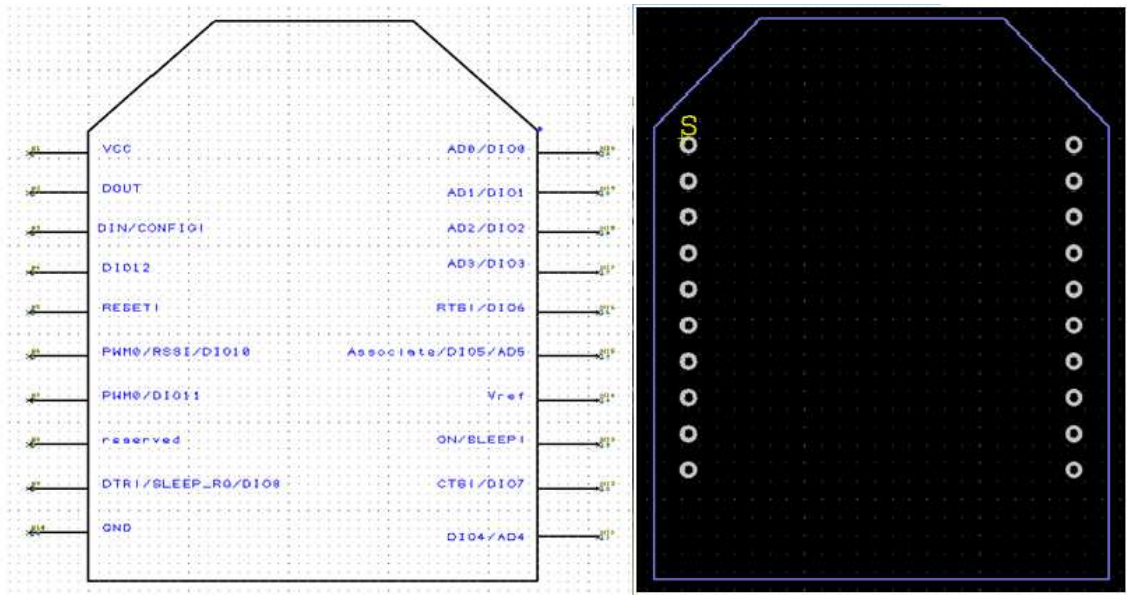


Figura 2.21 – Símbolo y footprint del módulo de radio

Una vez se tienen diseñados todos los componentes con sus correspondientes símbolos y huellas, se completa el esquemático del circuito. El siguiente paso es diseñar la placa de circuito impreso.

Para ello hay que seguir el siguiente proceso. Pinchar en la opción Tools → Translate to PCB. A continuación aparece una donde se debe seleccionar que tipo de PCB se desea diseñar (tecnología usada, número de capas, tamaño, etc) como se aprecia en la figura 2.22.

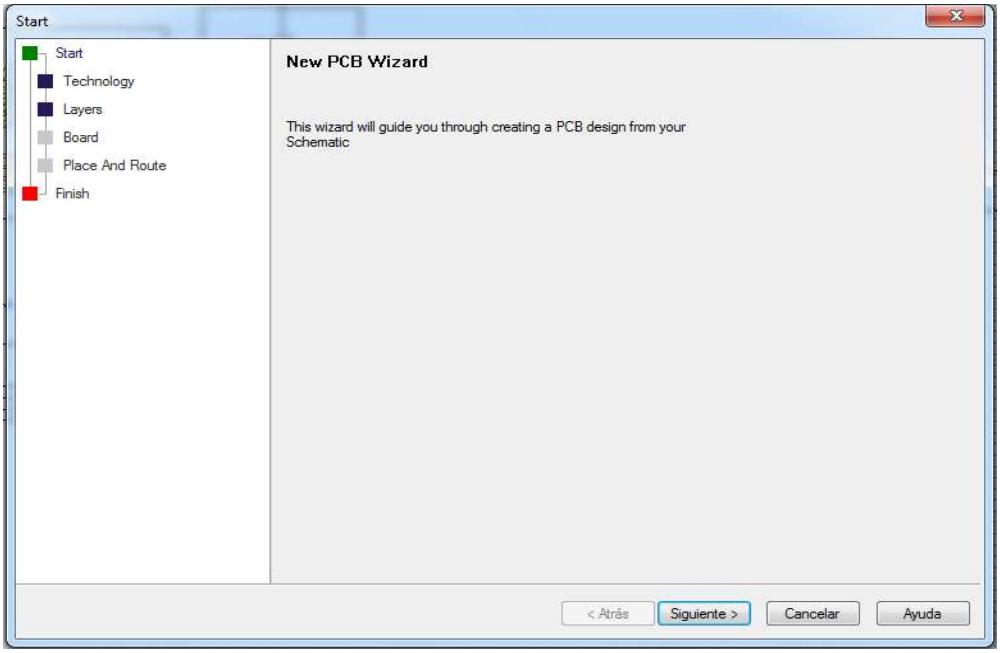


Figura 2.22 – Creación de una nueva PCB con DesignSpark

A continuación, en las figuras 2.23 y 2.24 se muestra el layout de las dos PCBs desarrolladas:

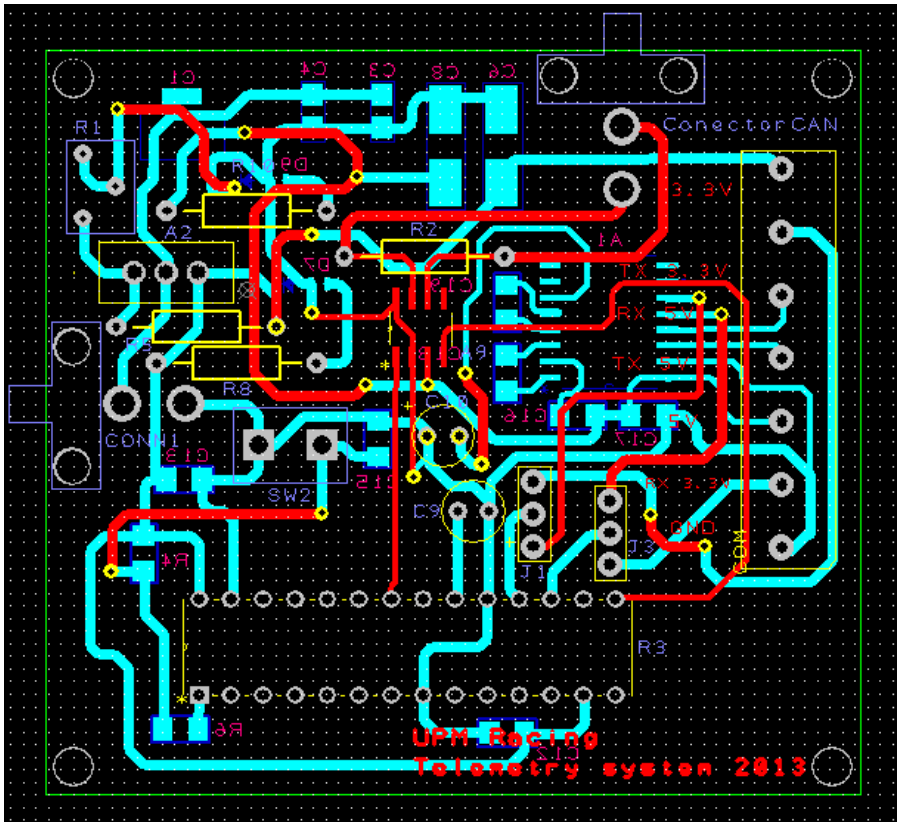


Figura 2.23 – Layout de la PCB del microprocesador

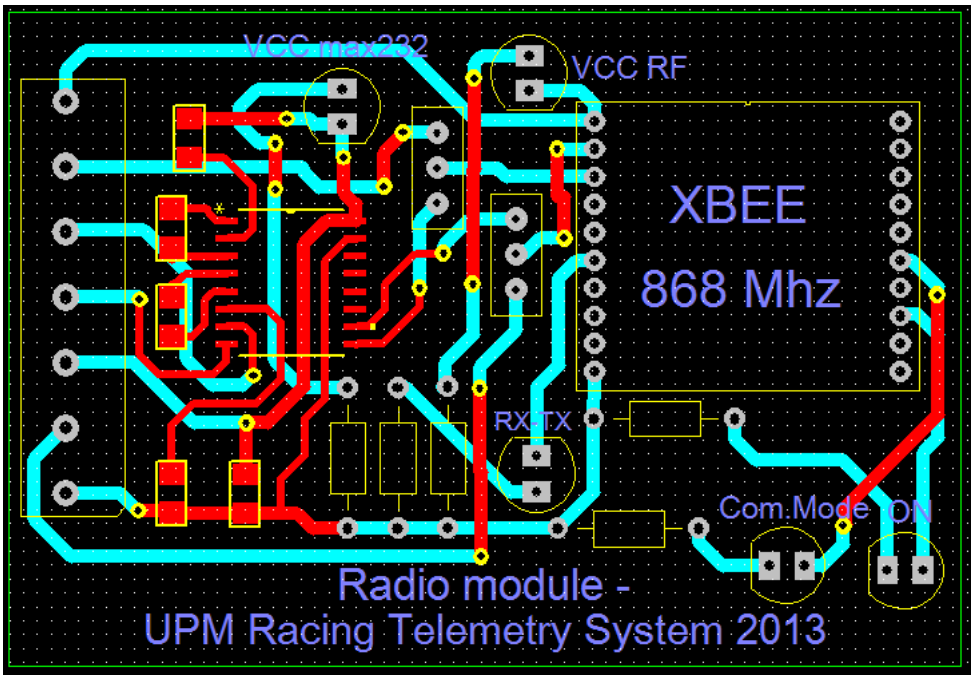


Figura 2.24 – Layout de la PCB del módulo de radio

2.9 FABRICACIÓN DE LAS PLACAS DE CIRCUITO IMPRESO

El método que se ha utilizado en este proyecto para realizar la placa PCB, una vez que está diseñada con los DesignSpark PCB es un método artesanal, ya que se basa en el uso de ácidos que permite, a partir de una placa de fibra de vidrio recubierta por una fina capa de cobre, conseguir un diseño a doble cara con buenos resultados.

El proceso comienza imprimiendo el diseño obtenido en el programa DesignSpark en un acetato o transparencia. Como se ha optado por realizar un diseño a doble cara es preciso imprimir por separado la parte superior (Top Layer) de la parte inferior (Bottom Layer). Entre ambas láminas de acetato se introduce la placa de fibra de vidrio con cobre en las dos caras. Se utiliza una placa PCB que es sensible a la luz ultravioleta.

Figura 2.25 - Imagen de la insoladora preparada para funcionar.

El paso siguiente consiste en introducir el conjunto de las transparencias impresas y la placa de fibra de vidrio en una insoladora que proyecta luz ultravioleta, la cual se muestra en la figura 2.25. En las partes de la placa PCB donde incida la luz proyectada por la insoladora el cobre se desprenderá con el proceso de atacado químico.

Los ácidos que se han de emplear van a realizar las funciones de revelador y atacante. El revelador se consigue diluyendo sosa cáustica en agua. La proporción empleada es 1 gramo de sosa cáustica por cada 100 ml de agua. En la figura 2.26 se muestran todos los componentes necesarios para llevar a cabo la fabricación de la PCB.

Figura 2.26 - Elementos necesarios para realizar una PCB.

El ácido atacador consiste en una mezcla de ácido clorhídrico, agua y agua oxigenada. Las proporciones son:

- 1 unidad de ácido clorhídrico (HCl).
- 1 unidad de agua oxigenada (H_2O_2).
- 2 unidades de agua (H_2O).

Como se puede apreciar en la figura 2.27 primero se introduce la PCB insolada en el revelador.

Figura 2.27 - Paso de la PCB por el revelador.

Una vez que el circuito queda claramente visible. La PCB ha de aclararse en agua (figura 2.28).

Figura 2.28 - Aclarado en agua tras el paso por el revelador.

A continuación es el turno del ácido atacante. Es preciso llevar cuidado sobretodo en esta parte del proceso ya que se liberan vapores tóxicos (figuras 2.29 y 2.30).

Figura 2.29 - Paso de la PCB por los ácidos.

Figura 2.30 - Paso de la PCB por los ácidos, vapores tóxicos.

Una vez que el cobre que no pertenece al circuito ha desaparecido se extrae la placa de los ácidos y se aclara en agua.

El paso siguiente consiste en la realización de los orificios pasantes utilizando un taladro vertical de altas revoluciones como el que se muestra en la figura 2.31. Una vez que estén todos los taladros realizados es preciso montar todos los componentes.

Figura 2.31 - Taladro.

En la figura 2.32 puede apreciarse el resultado final de la fabricación de ambas placas.

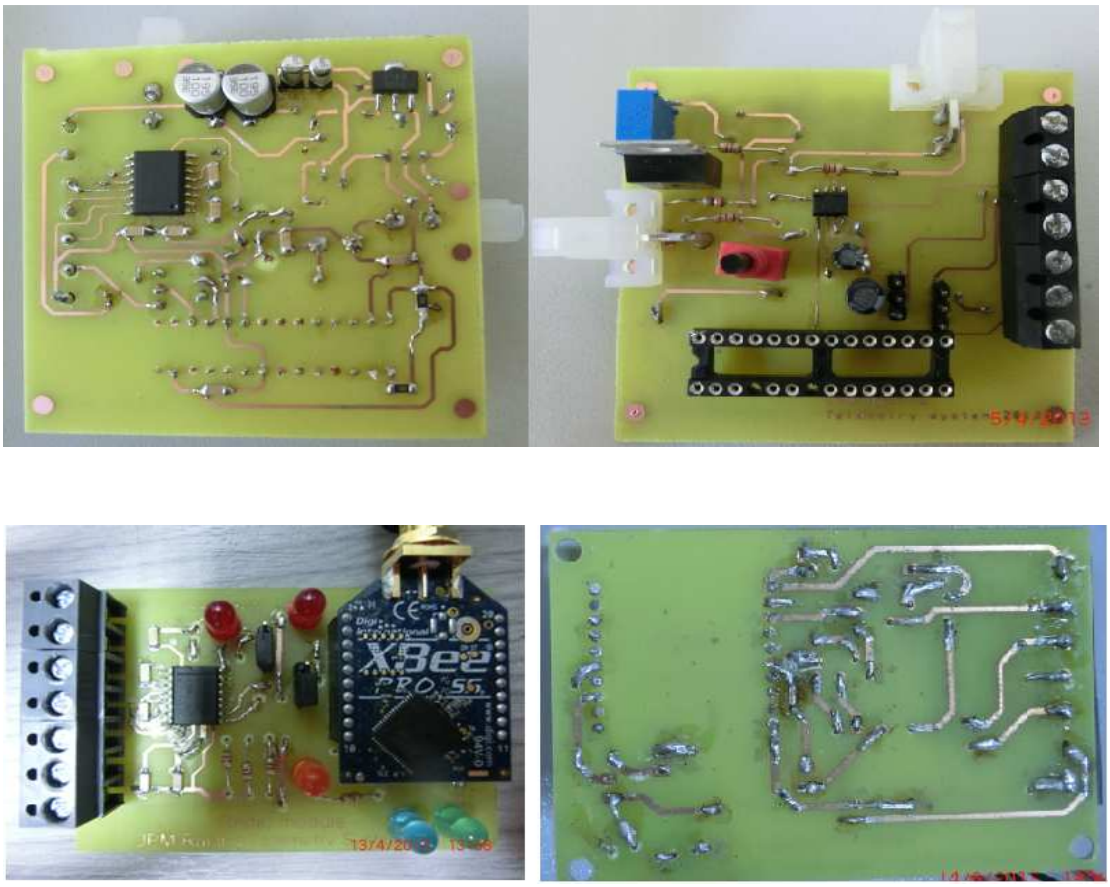


Figura 2.32 - Resultado final del montaje de las PCBs.

CAPITULO 3. DESARROLLO DEL SOFTWARE

3.1 INTRODUCCIÓN

El objetivo del software a desarrollar es recibir tramas de datos desde el bus CAN, las cuales nos indicarán el estado del vehículo, para posteriormente transmitir esos datos a través de la antena de radiofrecuencia. Para tal fin, se van a utilizar tres periféricos del microprocesador, que son: el módulo ECAN, para la comunicación con el bus CAN, el DMA para el almacenamiento de los mensajes CAN y la UART1 para la transmisión de los datos al chip de radio. Para el desarrollo de la aplicación se ha utilzado el entorno de desarrollo de microchip MPLAB, con su respectivo compilador para lenguaje C (C30).

3.2 ENTORNO DE DESARROLLO MPLAB

MPLAB IDE es un software para sistemas operativos windows, que sirve para desarrollar aplicaciones para los microprocesadores y controladores digitales de señal de microchip.

3.2.1 DESCRIPCIÓN DE UN SISTEMA EMBEBIDO

Un sistema integrado, empotrado o embebido es un sistema informático de uso específico construido dentro de un dispositivo mayor. Estos sistemas, típicamente, se desarrollan utilizando un microprocesador o controlador digital de señal. Los microprocesadores se componen de una unidad central de procesamiento o CPU, mas una serie de circuitos llamados periféricos, que junto a otros circuitos que hay dentro del chip forman un pequeño módulo de control. Este dispositivo único (microprocesador), puede ser integrado dentro de otros dispositivos electrónicos y mecánicos para aplicaciones de control digital con un bajo coste.

3.2.2 REALIZAR UN DISEÑO DE SISTEMA EMBEBIDO CON MPLAB IDE

Un sistema de desarrollo para controladores embebidos se compone de una serie de programas que se ejecutan en un PC, que ayudan a escribir, editar, depurar y programar código en un microprocesador, lo que se conoce como la inteligencia de los sistemas embebidos. Este conjunto de programas se conoce como la inteligencia de los sistemas embebidos. MPLAB IDE se ejecuta en un PC y contiene todo lo necesario para diseñar y utilizar aplicaciones de sistemas embebidos.

Las fases típicas en el desarrollo de una aplicación embebida son:

1. Crear un diseño de alto nivel. Una vez conocidas las especificaciones de la aplicación, hay que elegir el microprocesador (PIC MCU) o controlador digital de señal

(dsPIC DSC) que mejor se adapte a ellas y diseñar la circuitería asociada. Después de determinar que periféricos y pines controlan el hardware, se debe escribir el software. Para escribir y editar el código se puede usar lenguaje ensamblador, que es directamente traducible a código máquina (conjunto de unos y ceros) o bien un compilador que permite el uso de un lenguaje de programación más natural a la hora de crear el software

2. La segunda fase consiste por tanto en compilar, ensamblar y enlazar el software utilizando el ensamblador y/o compilador y enlazador para transformar el código en unos y ceros –código máquina para el microcontrolador PIC.

3. La siguiente etapa es testear o depurar el código. Para ello, se utiliza una herramienta llamada depurador, que permite ejecutar el código paso a paso y visualizar los valores que tienen las variables del programa en cada momento, para poder así corregir posibles errores de funcionamiento de la aplicación.

4. La última etapa del proceso es programar el código en el microprocesador y verificar que funciona correctamente en la aplicación final.

3.2.3 TUTORIAL DE MPLAB

A continuación se van a explicar las diferentes etapas que hay que seguir para desarrollar una aplicación con MPLAB IDE.

Seleccionar un dispositivo:

Lo primero de todo es seleccionar el dispositivo (microprocesador o controlador digital de señal que se va a utilizar). Para ello hay que entrar en el menú Configure y seleccionar la opción Configure→Device. Aparece una ventana como la que se muestra en la figura 3.1:

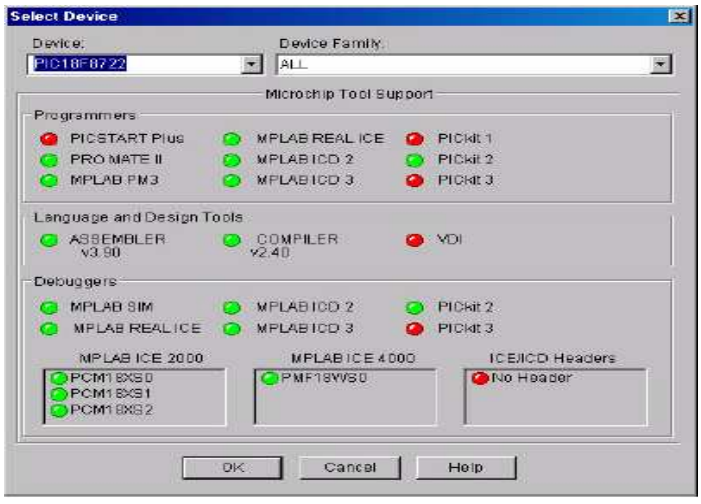


Figura 3.1 - Selección del dispositivo

Las luces indican qué componentes de MPLAB IDE soporta el dispositivo.

- Una luz verde indica soporte total.

- Una luz amarilla indica soporte preliminar para una herramienta de MPLAB IDE de próxima aparición.

- Una luz roja indica que el dispositivo elegido no soporta ese componente de MPLAB IDE.

Crear un proyecto:

La siguiente etapa es crear un proyecto utilizando la herramienta Project Wizard. Un proyecto es la manera en que se organizan los archivos para ser compilados y ensamblados. Para este proyecto se va a usar un único fichero ensamblador y un único enlazador. Pinchar en Project→Project Wizard y en la ventana de diálogo que aparece, pinchar en Next para avanzar como se puede ver en la figura 3.2.

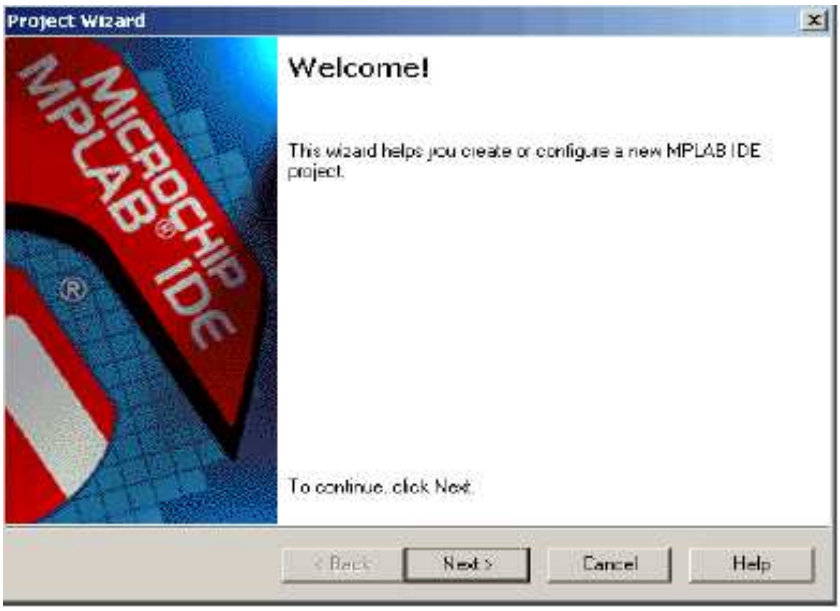


Figura 3.2 - Ventana de bienvenida del Project Wizard

Aparece a continuación una ventana donde hay que seleccionar el dispositivo. Si esto ya se ha hecho previamente, deberá aparecer el dispositivo seleccionado (figura 3.3).

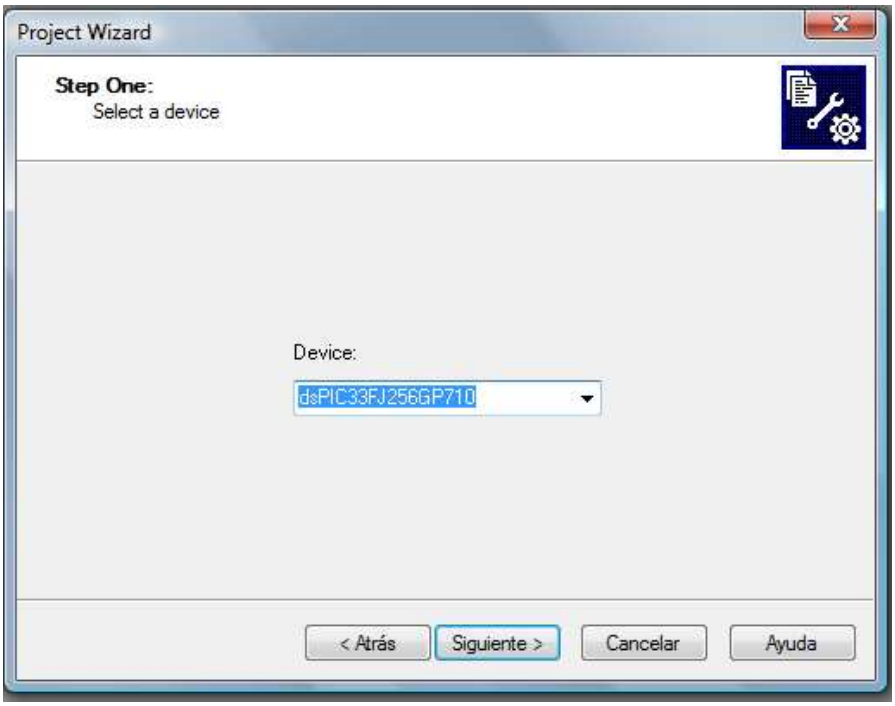


Figura 3.3 - Project Wizard - Selección del dispositivo

Establecer las herramientas del lenguaje:

La siguiente ventana del Project Wizard permite establecer las herramientas del lenguaje que se van a usar en este proyecto, como se muestra en la figura 3.4. En este caso se debe seleccionar la opción Microchip C30 Toolsuite. Esta herramienta es la que contiene el compilador y enlazador para lenguaje C. Al finalizar pinchar en Next.

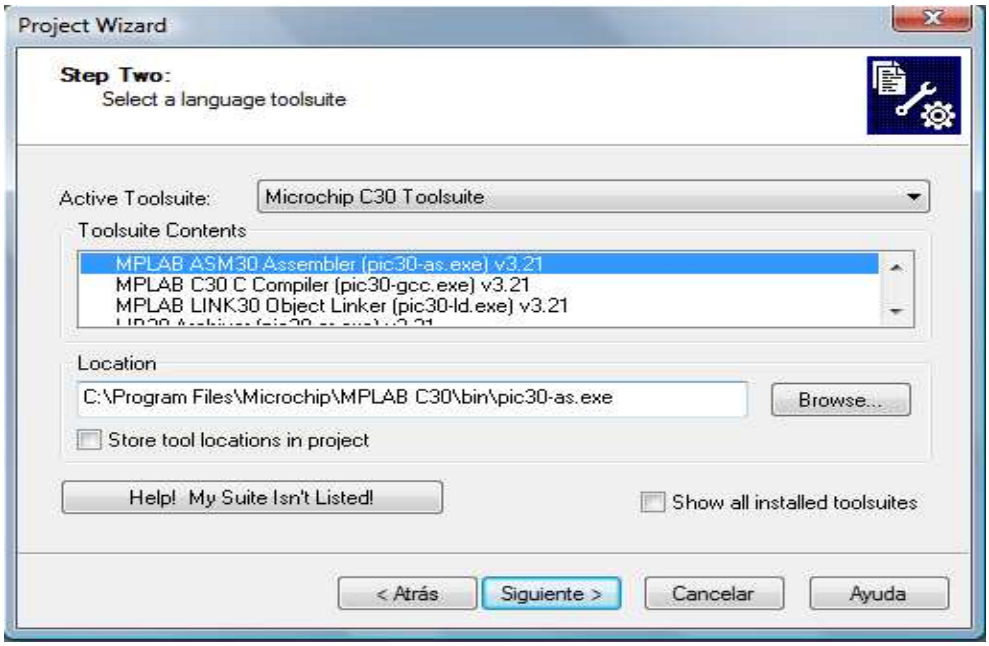


Figura 3.4 - Project Wizard - Selección de herramientas del lenguaje

Nombrar el proyecto:

En la siguiente ventana (figura 3.5) se debe dar nombre al proyecto y seleccionar su ubicación.

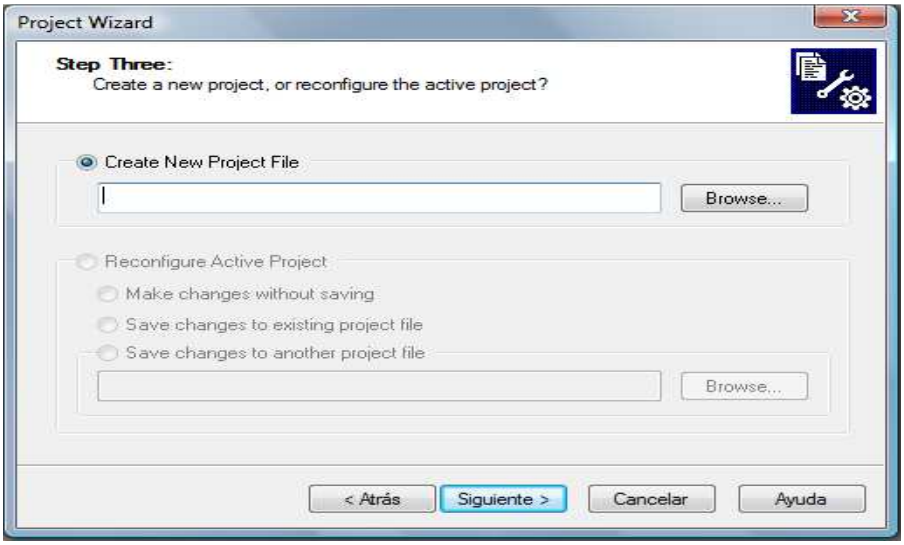


Figura 3.5 - Project Wizard - Nombre del proyecto

Añadir ficheros al proyecto:

La cuarta etapa del Project Wizard permite añadir ficheros al proyecto. Para empezar a escribir el programa se puede usar un fichero template. Un fichero template, es un fichero que contiene las secciones esenciales para cualquier fichero fuente y contiene información que puede ayudar a escribir y organizar el código. En la figura 3.6 se puede ver el aspecto de dicha etapa del Project Wizard.

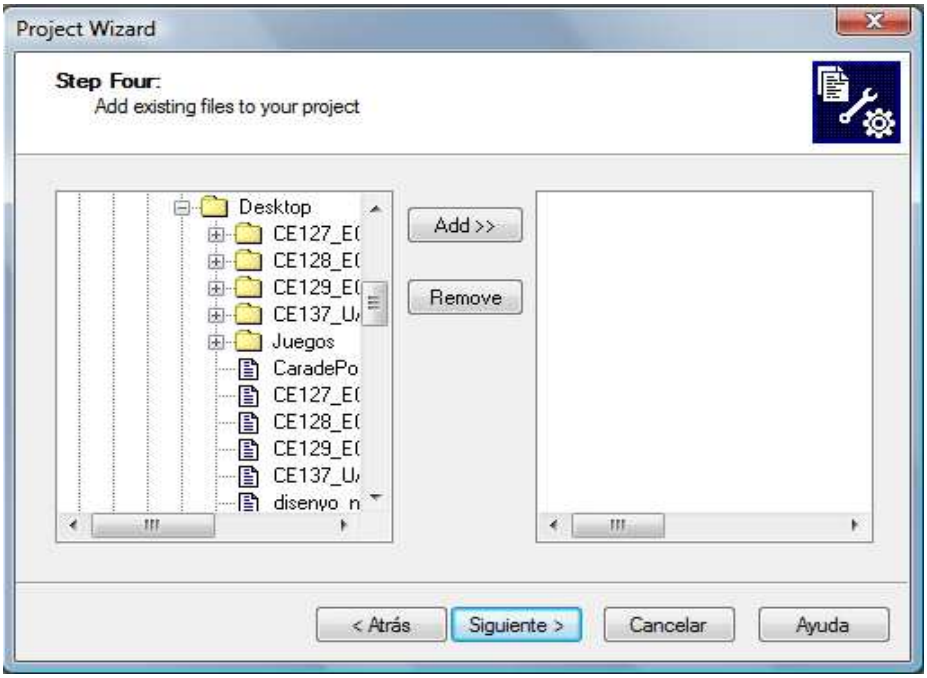


Figura 3.6 – Project Wizard – Seleccionar fichero template

La última ventana del Project Wizard es una ventana resumen, donde se indica el dispositivo seleccionado, las herramientas del lenguaje elegidas y el nombre del nuevo proyecto, como puede apreciarse en la figura 3.7.

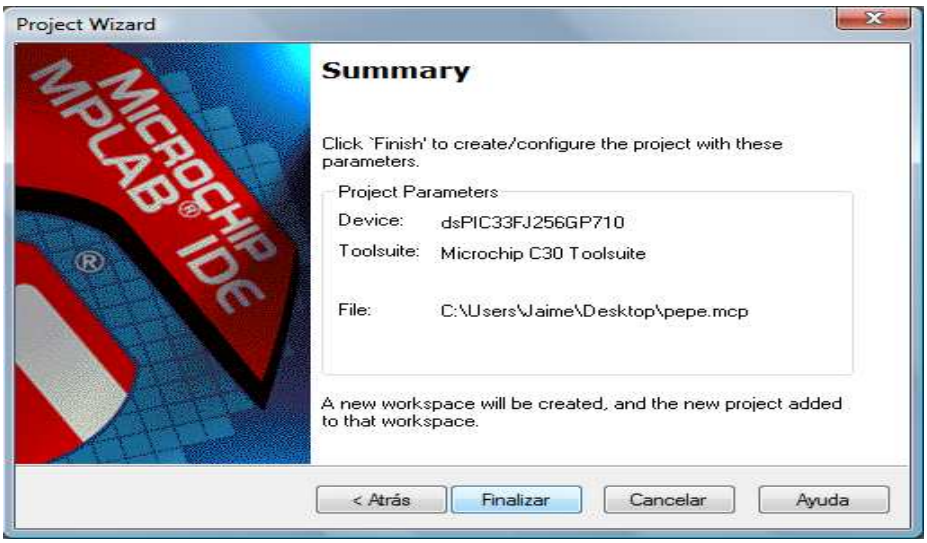


Figura 3.7 – Project Wizard - Resumen

Escribir el código:

Si se utiliza un fichero template como soporte para escribir el código, se observará que este tiene diferentes secciones. Al principio del fichero hay una zona de comentarios donde se puede escribir información específica del proyecto que se vaya a realizar. Posteriormente, hay una zona reservada a funciones especiales, como pueden ser las rutinas de atención a las interrupciones o las macros de los bits de configuración. A continuación se encuentra la función principal del programa.

Compilar el proyecto:

Seleccionar Project→Build All para ensamblar y enlazar el código. Si el código tiene algún error, se deberá corregir. Para ello, al hacer doble clic sobre el mensaje de error, MPLAB IDE señalará la línea del código fuente donde se encuentra el error. Una vez corregido el error se debe volver a compilar el proyecto. Este proceso se debe repetir hasta corregir todos los errores.

Depurar el código:

Para poder depurar el código sin tener que utilizar dispositivos reales, MPLAB IDE posee una herramienta como es el simulador MPLAB SIM. El simulador es un software que se ejecuta en un PC y simula la ejecución de las instrucciones del microprocesador. Para arrancar el simulador, pinchar en el menú Debugger→Select Tool y seleccionar MPLAB SIM como opción de depuración, como se muestra en la figura 3.8.

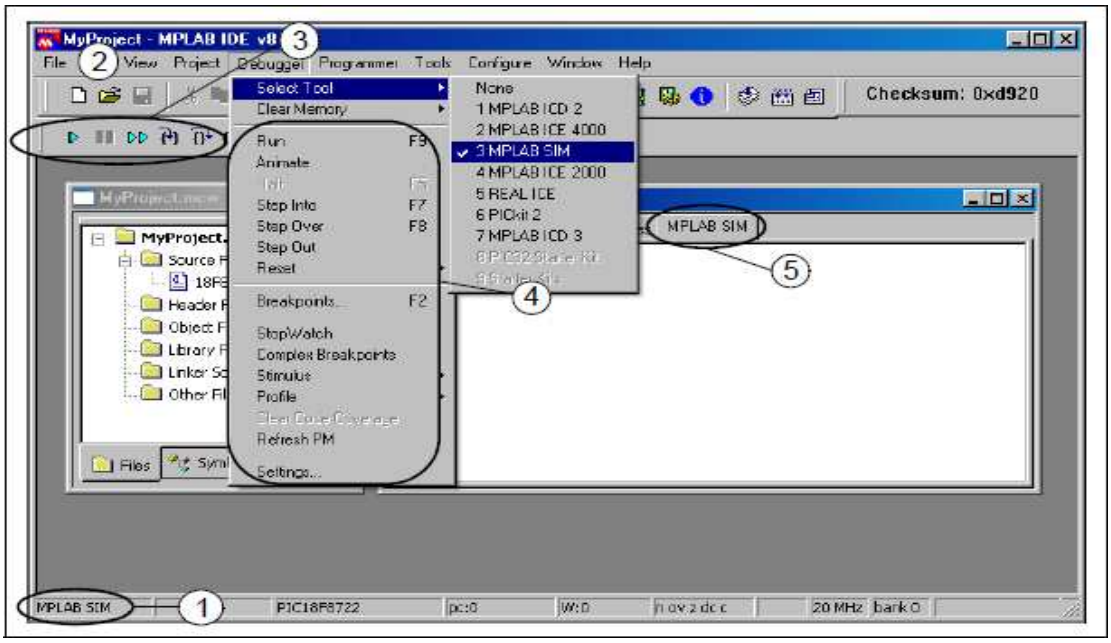


Figura 3.8 - Selección de la herramienta de depuración

El depurador tiene una serie de herramientas para la depuración del código, las cuales se muestran en la siguiente tabla:

Tabla 15 - Menú del depurador

Debugger Menu	Toolbar Buttons	Hot Key
Run		F9
Halt		F5
Animate		
Step Into		F7
Step Over		F8
Step Out		
Reset		F6

La opción Run hace que el código empiece a ejecutarse.

La opción Halt detiene la ejecución del código.

Con la opción Animate se ejecuta el código a la vez que el diseñador puede visualizar qué parte de código se está ejecutando en cada momento.

La opción Step Into sirve para depurar paso a paso. Con esta opción avanzamos una línea de código.

La opción Step over sirve para ejecutar funciones sin tener que hacerlo paso a paso.

Otra herramienta muy útil del depurador es la ventana para visualizar variables y registros (Watch Window). Esta ventana, que se muetra en la figura 3.10, tiene dos menús, uno para seleccionar los registros SFR de los que se quieren visualizar sus valores y otro para añadir las variables.

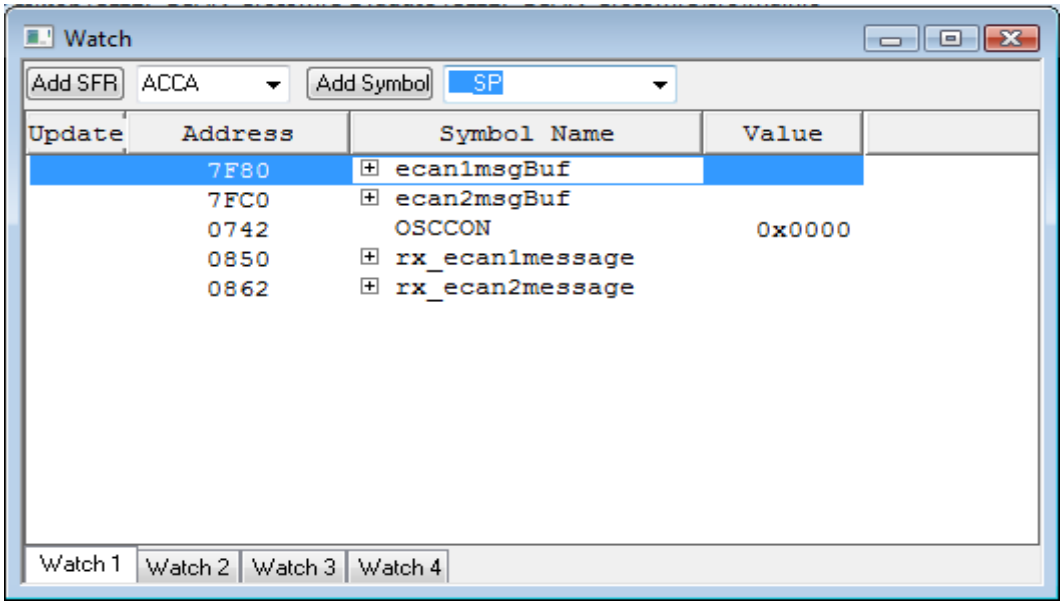


Figura 3.10 - Watch Window

Otra opción muy útil a la hora de depurar un programa, es utilizar puntos de ruptura. El punto de ruptura marca un punto del programa donde se detendrá la ejecución del mismo, pudiéndose observar así el valor de las variables o registros que interese en ese momento. Para colocar un punto de ruptura hay dos opciones. La primera es hacer doble clic con el ratón en la línea donde se quiere situar el punto de ruptura, y la segunda es, una vez situado el cursor del ratón en la línea donde se quiere colocar el punto de ruptura, se pulsa el botón derecho del ratón, y en el menú que se despliega se selecciona la opción Set Breakpoint.

3.3 PLACA DE DESARROLLO MICROSTICK

La tarjeta Microstick es una tarjeta de desarrollo de Microchip para los microprocesadores y controladores digitales de señal de 16 bits PIC24H y dsPIC33F. La tarjeta tiene un programador/depurador integrado, un zócalo para conectar el dispositivo bajo prueba y dos tiras de pines que permiten conectar la tarjeta en una protoboard y proporcionan una gran flexibilidad a la hora de desarrollar aplicaciones.

La tarjeta es soportada por la plataforma de desarrollo MPLAB. Las características más destacables de la Microstick son:

- Programador/depurador integrado. No se requiere depurador externo.
- Alimentación a través de USB. No requiere alimentación externa.

- Zócalo integrado. Fácil sustitución del dispositivo.
- LED indicador de modo depuración.
- Interruptor de Reset.

3.4 OSCILADOR

El sistema oscilador del microprocesador PIC24HJ64GP502 proporciona:

- Varias fuentes de reloj, tanto internas como externas.
- Un PLL integrado en el chip para ajustar la frecuencia de funcionamiento interna del microprocesador a la frecuencia requerida por el sistema.
- El oscilador interno FRC puede utilizar también el PLL, permitiendo así, el funcionamiento del sistema a altas velocidades, sin la necesidad de ningún reloj externo.
- Conmutación entre varias fuentes de reloj.
- Un Fail-Safe Clock Monitor (FSCM) que detecta fallos en el reloj y toma muestras.
- Un registro de control (OSCCON).
- Bits de configuración para seleccionar el oscilador principal.

A continuación, en la figura 3.11 se muestra un diagrama del sistema oscilador:

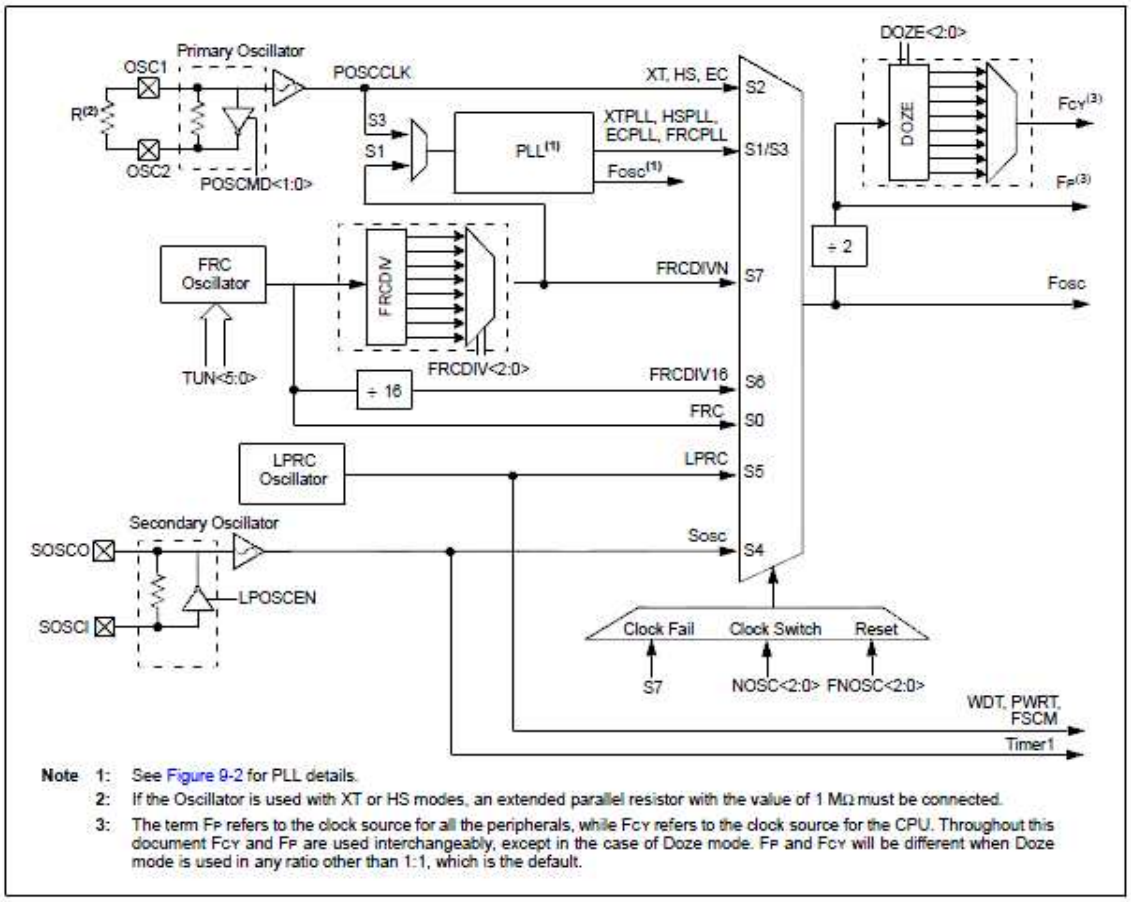


Figura 3.11 - Sistema oscilador [5]

Como se aprecia en la figura anterior, existen siete posibles fuentes de reloj:

- Oscilador FRC.
- Oscilador FRC con PLL.
- Oscilador primario (XT, HS o EC).
- Oscilador primario con PLL.
- Oscilador secundario (LP).
- Oscilador LPRC.
- Oscilador FRC con postscaler.

Para el presente proyecto, se va a utilizar el oscilador interno, ya que simplifica el hardware del sistema.

El oscilador interno FRC, genera una señal de frecuencia nominal 7.37MHz. Esta frecuencia se puede modificar por software, en un rango desde -12% hasta 11,625%, escribiendo el valor adecuado en el registro OSCTUN.

3.4.1 SELECCION DEL RELOJ DEL SISTEMA

La fuente de reloj que es utilizada tras el reset del dispositivo, se selecciona mediante los bits de configuración. Los bits de configuración para la selección del oscilador inicial, FNOSC <2:0> que se encuentran en el registro (FOSCSEL <2:0>) y los bits de configuración para la selección del modo de funcionamiento del oscilador primario, POSCMD<1:0>, que se encuentran en el registro (FOSC<1:0>), seleccionan la fuente de reloj que se utilizará tras el reset. Se puede elegir entre doce modos de reloj distintos, tal como muestra la figura 3.12.

Oscillator Mode	Oscillator Source	POSCMD<1:0>	FNOSC<2:0>	Note
Fast RC Oscillator with Divide-by-N (FRCDIVN)	Internal	xx	111	1, 2
Fast RC Oscillator with Divide-by-16 (FRCDIV16)	Internal	xx	110	1
Low-Power RC Oscillator (LPRC)	Internal	xx	101	1
Secondary (Timer1) Oscillator (SOSC)	Secondary	xx	100	1
Primary Oscillator (HS) with PLL (HSPLL)	Primary	10	011	—
Primary Oscillator (XT) with PLL (XTPLL)	Primary	01	011	—
Primary Oscillator (EC) with PLL (ECPLL)	Primary	00	011	1
Primary Oscillator (HS)	Primary	10	010	—
Primary Oscillator (XT)	Primary	01	010	—
Primary Oscillator (EC)	Primary	00	010	1
Fast RC Oscillator with PLL (FRCPLL)	Internal	xx	001	1
Fast RC Oscillator (FRC)	Internal	xx	000	1

Figura 3.12 - Valores de los bits de configuración para la selección del reloj [5]

3.4.2 CONFIGURACION DEL PLL

El oscilador primario y el oscilador interno FRC, pueden usar el PLL para obtener mayores velocidades de funcionamiento. El PLL proporciona una considerable flexibilidad a la hora de seleccionar la velocidad de funcionamiento del microprocesador. A continuación, se muestra el diagrama de bloques del PLL y se explica su funcionamiento (figura 3.13).

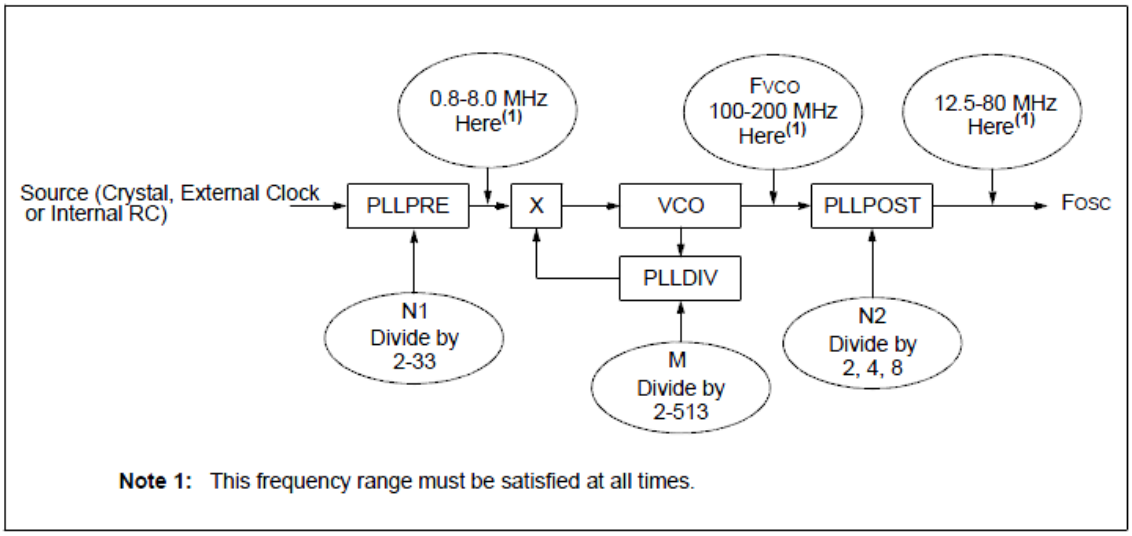


Figura 3.13 - Diagrama de bloques del PLL [5]

La salida del oscilador primario o el oscilador FRC, denominada FIN, se divide por un factor (N1) de 2, 3, 4....33, antes de atacar al VCO (oscilador controlado por tensión) del PLL. La entrada del VCO debe estar en el rango entre 0.8MHz y 8MHz. El factor de preescalado N1 se selecciona mediante los bits PLLPRE<4:0> del registro (CLKDIV<4:0>). El divisor de realimentación del PLL (PLLDIV), seleccionado mediante los bits PLLDIV<8:0> del registro (PLLFBD<8:0>), proporciona un factor M, que multiplica la entrada del VCO. Este factor debe escogerse de tal manera que, la frecuencia de salida del VCO esté entre 100MHz y 200MHz. Por último, la salida del VCO es dividida por un factor de postescalado (N2). Este factor de postescalado se selecciona mediante los bits PLLPOST<1:0> del registro (CLKDIV<7:6>). N2 puede ser igual a 2, 4 u 8 y debe escogerse de tal manera que la salida del PLL (Fosc) esté entre 12.5MHz y 80MHz. La ecuación que permite calcular la frecuencia de salida del PLL se muestra en la ecuación (1):

$$Fosc = FIN \cdot \left(\frac{M}{N1 \cdot N2}\right) \qquad (1)$$

Figura 3.14 - Ecuación para el cálculo de la frecuencia de salida del PLL

Este es el código desarrollado para la configuración del oscilador del microprocesador:

Macros para seleccionar la fuente de reloj

```
_FOSCSEL(FNOSC_FRC & IESO_OFF);  // Select Internal FRC at POR, Two-speed Oscillator Startup
                                 //Disabled
_FOSC(FCKSM_CSECMD);        // Clock Switching is enabled and Fail Safe Clock Monitor is disabled



/******************************************************************************
*   Funcion:          oscConfig
*   Description:      Configura el oscilador del sistema
*
*   Parametros:       Ninguno
*
*   Autor:            Jaime Iglesias
******************************************************************************/
void oscConfig(void)
{

        /*  Configure Oscillator to operate the device at 40Mhz
                Fosc= Fin*M/(N1*N2), Fcy=Fosc/2
                Fosc= 8M*40/(2*2)=80Mhz for 8M input clock */

                PLLFBD=38;                      /* M=40 */
                CLKDIVbits.PLLPOST=0;           /* N1=2 */
                CLKDIVbits.PLLPRE=0;            /* N2=2 */

        /*      Write to OSCTUN the appropiate value to adjust the frequency
                of FRC oscillator at 8MHz.
                Write 1 increment the center frequency(7.37 MHz) an 0.375%
```

```
        Writing 22 we achieve a frequency of 7.978 MHz
        Writing 23 we achieve a frequency of 8.0056 MHz*/

        OSCTUN=0x0017;                  /* Tune FRC oscillator, if FRC is used */

    /* Clock switch to incorporate PLL*/
        __builtin_write_OSCCONH(0x01);          // Initiate Clock Switch to Primary
        //FRC Oscillator with PLL (NOSC=0b001)
        __builtin_write_OSCCONL(0x01);      // Start clock switching
        while (OSCCONbits.COSC != 0b001);   // Wait for Clock switch to occur

    /* Wait for PLL to lock */
        while (OSCCONbits.LOCK!=1) {};
}
```

3.5 CONFIGURACION DE PINES DE ENTRADA/SALIDA

El microprocesador posee hasta 85 pines digitales de entrada/salida programables. Todos los pines del microprocesador (excepto VDD, VSS, MCLR y OSC1/CLKIN), están compartidos entre los periféricos y los puertos paralelos de entrada/salida.

3.5.1 PUERTOS PARALELOS DE ENTRADA/SALIDA

Un puerto paralelo de entrada/salida que comparte un pin con un periférco, es en general subordinado del periférico. Las señales de control y los datos de salida del buffer del periférico, son proporcionados por un par de multiplexores. Los multiplexores seleccionan quién tiene el control de los datos de salida y las señales de control del pin de entrada/salida, si el periférico o el puerto asociado. La figura 3.15 ilustra cómo están compartidos los puertos con otros periféricos y los pines de entrada/salida a los cuales están conectados.

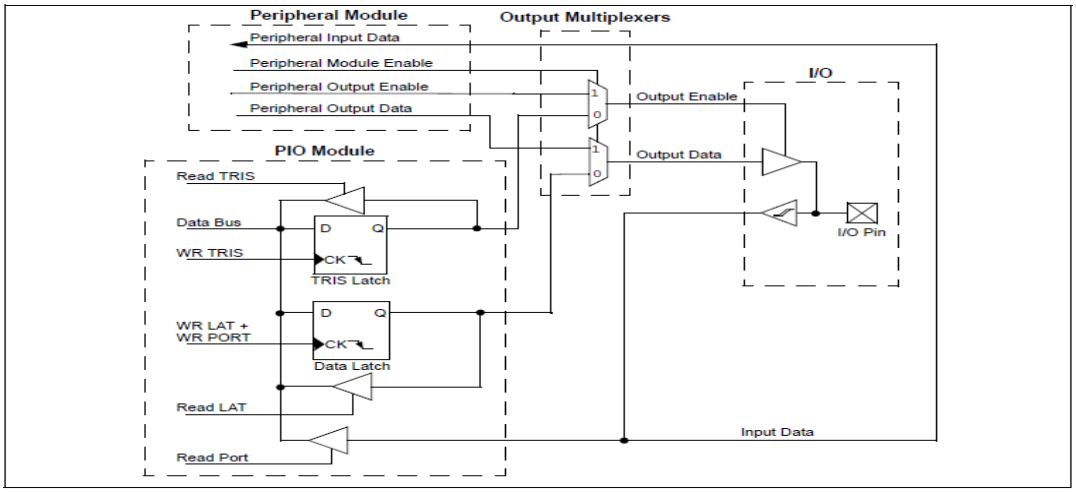


Figura 3.15 - Diagrama de bloques de una típica estructura compartida de puerto [5]

Todos los pines del puerto tienen tres registros directamente asociados con su funcionamiento como entrada/salida digital. El registro de dirección de datos (TRISx) determina si el pin es una entrada o una salida. Si el bit de dirección de datos es un '1', el pin se configura como entrada, mientras que si es un '0' se configura como salida. Todos los pines están configurados como entradas tras el reset del microprocesador. Las lecturas del latch (LATx) leen el latch. Las escrituras en el latch, escriben el latch. Las lecturas del puerto (PORTx) leen los valores de los pines del puerto, mientras que las escrituras del puerto, escriben el latch. A continuación, se muestra la función de configuración de los pines de E/S utilizados en el proyecto.

Una característica muy interesante del microprocesador que se ha utilizado es que permite elegir en que pines estarán disponibles ciertos periféricos. Hay hasta 26 pines disponibles remapeables dependiendo del dispositivo (denominados con RPn en la asignación de pines).

Esta característica del microprocesador se controla mediante dos registros de función especial (SFR), uno para mapear las entradas (RPINRx, x=0..25) y otro para mapear las salidas (RPORx). Por comodidad a la hora de rutear la PCB del microprocesador se han asignado las señales CAN a los pines 15 (RP6 (RXD)) y 22 (RP11 (TXD)) y las señales de la UART a los pines 18 (RP9 (UART TX)) y 17 (RP8 (UART RX)).

A continuación se muestra el código de configuración de los pines de E/S:

```
/*********************************************************************************
*   Funcion:                    ioPinConfig
*   Descripcion:      configura los pines de E/S del micro que se van a usar en la aplicación
*
*  Parametros:              Ninguno
*  Author:          Jaime Iglesias
*********************************************************************************/
void ioPinConfig (void)
{
        //Tras el reset todos los pines están configurados como entradas
        //Bit del correspondiente registro TRISx a '1'. Bit a '0'configura el puerto como salida
        TRISB = 0xF5FF;             //RB11, RB9 outputs.
        RPINR26 = 0x0006; //ECAN1 Receive to RP6 pin
        RPOR5 = 0x1000;   //ECAN1 Transmit to RP11
        RPOR4 = 0x0300;   // UART1 TX to RP9
        RPINR18 = 0x0008;   // UART1 RX to RP8
}
```

3.6 UART: REGISTROS Y CONFIGURACION

El módulo UART (Universal Asynchronous Receiver Transmitter) es uno de los módulos serie de entrada/salida de que disponen los dispositivos de la familia PIC24H. La UART es un canal de comunicación asíncrono full-duplex que se comunica con periféricos y ordenadores personales, usando protocolos como el RS-232, RS-485, LIN e IrDA. Este módulo también soporta el control de flujo hardware mediante los pines UxCTS y UxRTS, e incluye el codificador y decodificador IrDA.

Las principales características del módulo UART son las siguientes:

- Transmisión de datos de 8 o 9 bits full duplex a través de los pines UxTX y UxRX.

- Opciones de paridad par, impar o no paridad (para datos de 8 bit).
- Uno o dos bits de parada.
- Opción de control de flujo hardware a través de los pines UxCTS y UxRTS.
- Generador de baud-rate con prescaler de 16 bits totalmente integrado.
- Velocidades desde 10Mbps hasta 38bps a 40 MIPS.
- Buffer de datos transmitidos FIFO (First-In, First-Out) de 4 posiciones.
- Buffer de datos recibidos FIFO de 4 posiciones.
- Detección de error de desbordamiento de buffer, paridad y encuadrado.
- Soporta el modo de 9 bits con detección de dirección.
- Interrupciones de transmisión y recepción.
- Lógica del codificador y decodificador IrDA.
- Soporte para el bus LIN.

En la figura 3.16 se muestra un diagrama de bloques simplificado de la UART:

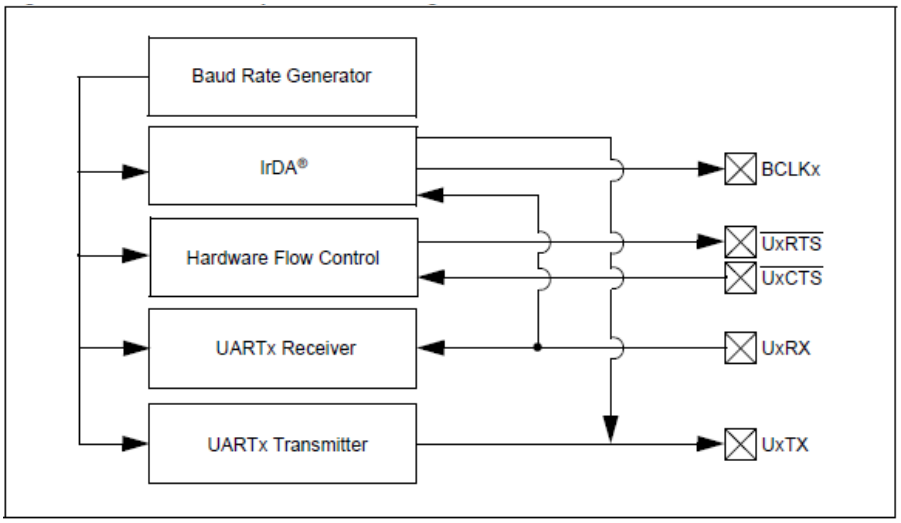


Figura 3.16 - Diagrama de bloques de la UART [5]

A continuación se describen los registros de control de la UART:

REGISTRO UxMODE

En este registro se habilitan y deshabilitan la UART, el codificador y decodificador IrDA, las opciones WAKE, ABAUD y Loopback y los pines UxRTS y UxCTS. También se puede configurar el pin UxRTS en el modo de funcionamiento deseado, la polaridad del pin UxRX, así como seleccionar el baud rate o el número de bits de datos y los bits de paridad y de parada. En la figura 3.17 se muestran los bits del registro UxMODE.

Register 17-1: UxMODE: UARTx Mode Register

R/W-0	U-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
UARTEN	—	USIDL	IREN(1)	RTSMD	—	UEN<1:0>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WAKE	LPBACK	ABAUD	URXINV	BRGH	PDSEL<1:0>		STSEL
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15 **UARTEN:** UARTx Enable bit
1 = UARTx is enabled; UARTx pins are controlled by UARTx as defined by the UEN<1:0> and UTXEN control bits
0 = UARTx is disabled; UARTx pins are controlled by the corresponding PORT, LAT and TRIS bits

bit 14 **Reserved**

bit 13 **USIDL:** Stop in Idle Mode bit
1 = Discontinue operation when the device enters Idle mode
0 = Continue operation in Idle mode

bit 12 **IREN:** IrDA Encoder and Decoder Enable bit(1)
1 = IrDA encoder and decoder are enabled
0 = IrDA encoder and decoder are disabled

bit 11 **RTSMD:** Mode Selection for $\overline{\text{UxRTS}}$ Pin bit
1 = $\overline{\text{UxRTS}}$ is in Simplex mode
0 = $\overline{\text{UxRTS}}$ is in Flow Control mode

bit 10 **Reserved**

bit 9-8 **UEN<1:0>:** UARTx Enable bits
11 = UxTX, UxRX and BCLKx pins are enabled and used; $\overline{\text{UxCTS}}$ pin is controlled by port latches
10 = UxTX, UxRX, $\overline{\text{UxCTS}}$ and $\overline{\text{UxRTS}}$ pins are enabled and used
01 = UxTX, UxRX and $\overline{\text{UxRTS}}$ pins are enabled and used; $\overline{\text{UxCTS}}$ pin is controlled by port latches
00 = UxTX and UxRX pins are enabled and used; $\overline{\text{UxCTS}}$, $\overline{\text{UxRTS}}$ and BCLKx pins are controlled by port latches

bit 7 **WAKE:** Enable Wake-up on Start bit Detect During Sleep Mode bit
1 = Wake-up is enabled
0 = Wake-up is disabled

bit 6 **LPBACK:** UARTx Loopback Mode Select bit
1 = Enable Loopback mode
0 = Loopback mode is disabled

bit 5 **ABAUD:** Auto-Baud Enable bit
1 = Enable baud rate measurement on the next character. Requires reception of a Sync field (55h); cleared in hardware upon completion.
0 = Baud rate measurement disabled or completed

bit 4 **URXINV:** Receive Polarity Inversion bit
1 = UxRX Idle state is '0'
0 = UxRX Idle state is '1'

Note 1: This feature is only available for Low-Speed mode (BRGH = 0). See the specific device data sheet for details.

Figura 3.17 – Registro UxMODE [5]

REGISTRO UxSTA

Este es el registro de control y estado de la UART. En él se puede seleccionar el modo de la interrupción de transmisión y de recepción, habilitar o deshabilitar la transmisión

de la UART, controlar el modo de detección de dirección e indicar varias condiciones de estado, como pueden ser, el estado del buffer de transmisión y recepción, o los errores de paridad y desbordamiento. En la figura 3.18 se muestran los bits del registro UxSTA.

Register 17-2: UxSTA: UARTx Status and Control Register

R/W-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0	R-0	R-1
UTXISEL1	UTXINV	UTXISEL0	—	UTXBRK	UTXEN	UTXBF	TRMT
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R-1	R-0	R-0	R/C-0	R-0
URXISEL<1:0>		ADDEN	RIDLE	PERR	FERR	OERR	URXDA
bit 7							bit 0

Legend:	C = Clearable bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15,13 **UTXISEL<1:0>** Transmission Interrupt Mode Selection bits
11 = Reserved
10 = Interrupt generated when a character is transferred to the Transmit Shift register and the transmit buffer becomes empty
01 = Interrupt generated when the last transmission is over (i.e., the last character has been shifted out of the Transmit Shift register) and all the transmit operations are completed
00 = Interrupt generated when any character is transferred to the Transmit Shift Register (which implies at least one location is empty in the transmit buffer)

bit 14 **UTXINV:** Transmit Polarity Inversion bit
1 = UxTX Idle state is '1'
0 = UxTX Idle state is '0'

bit 12 **Unimplemented:** Read as '0'

bit 11 **UTXBRK:** Transmit Break bit
1 = UxTX pin is driven low regardless of the transmitter state (Sync Break transmission – Start bit followed by twelve '0's and a Stop bit)
0 = Sync Break transmission is disabled or completed

bit 10 **UTXEN:** Transmit Enable bit
1 = UARTx transmitter enabled; UxTX pin is controlled by UARTx (if UARTEN = 1)
0 = UARTx transmitter disabled; any pending transmission is aborted and the buffer is reset; UxTX pin is controlled by PORT

bit 9 **UTXBF:** Transmit Buffer Full Status bit (read-only)
1 = Transmit buffer is full
0 = Transmit buffer is not full; at least one more data word can be written

bit 8 **TRMT:** Transmit Shift Register is Empty bit (read-only)
1 = Transmit Shift register is empty and the transmit buffer is empty (i.e., the last transmission has completed)
0 = Transmit Shift register is not empty; a transmission is in progress or queued in the transmit buffer

bit 7-6 **URXISEL<1:0>:** Receive Interrupt Mode Selection bits
11 = Interrupt flag bit is set when the receive buffer is full (i.e., has 4 data characters)
10 = Interrupt flag bit is set when the receive buffer is 3/4 full (i.e., has 3 data characters)
0x = Interrupt flag bit is set when a character is received

bit 5 **ADDEN:** Address Character Detect bit (bit 8 of received data = 1)
1 = Address Detect mode enabled. If 9-bit mode is not selected, this control bit has no effect.
0 = Address Detect mode disabled

bit 4 **RIDLE:** Receiver Idle bit (read-only)
1 = Receiver is Idle
0 = Data is being received

Figura 3.18 – Registro UxSTA [5]

REGISTRO UxRXREG

Este registro almacena los datos recibidos por la UART. En la figura 3.19 se muestran los bits del registro UxRXREG.

Register 17-3: UxRXREG: UARTx Receive Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R-0
—	—	—	—	—	—	—	URX8
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
URX<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-9 **Unimplemented:** Read as '0'
bit 8 **URX8:** Data bit 8 of the Received Character (in 9-bit mode)
bit 7-0 **URX<7:0>:** Data bits 7-0 of the Received Character

Figura 3.19 – Registro UxRXREG [5]

REGISTRO UxTXREG

Este registro almacena los datos que van a ser transmitidos. En la figura 3.20 se muestran los bits del registro UxTXREG.

Register 17-4: UxTXREG: UARTx Transmit Register (Write-Only)

U-0	U-0	U-0	U-0	U-0	U-0	U-0	W-x
—	—	—	—	—	—	—	UTX8
bit 15							bit 8

W-x	W-x	W-x	W-x	W-x	W-x	W-x	W-x
UTX<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-9 **Unimplemented:** Read as '0'
bit 8 **UTX8:** Data bit 8 of the Transmitted Character (in 9-bit mode)
bit 7-0 **UTX<7:0>:** Data bits 7-0 of the Transmitted Character

Figura 3.20 - Registro UxTXREG [5]

REGISTRO UxBRG

El registro UxBRG almacena el valor del baud rate del dato que se transmite o se recibe. A continuación se exponen los cálculos realizados con el bit BRGH = 0 para un baudrate de 19200 baudios. El bit BRGH, presente en el registro UxMODE, permite seleccionar si se quiere configurar la UART con alta velocidad o no. Para seleccionar alta velocidad en la UART hay que configurar el bit BRGH = 1. En la figura 3.21 se muestran los bits del registro UxBRG y en la ecuación (2) aparecen los cálculos del valor a cargar en el registro para obtener la velocidad de la UART deseada.

$$BaudRate = \frac{Fcy}{16x(UxBRG+1)} \quad \text{--> } \quad UxBRG = \frac{Fcy}{16xBaudRate} - 1 = \frac{40MHz}{16x9600} - 1 = 129 \qquad (2)$$

Register 17-5: UxBRG: UARTx Baud Rate Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	W-x
BRG<15:8>							
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
BRG<7:0>							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-0 **BRG<15:0>:** Baud Rate Divisor bits

Figura 3.21 - Registro UxBRG [5]

Este es el código en C que realiza la configuración de la UART1

```
void UART1_Config(void)
{
    /************************UART 1 Configuration*************************************
    *
    *           9600 baudrate
    *           Low baud rate
    *           8 bit transmission/reception
    *           No parity bit
    *           1 stop bit
    *
    ********************************************************************************/
```

```
    U1MODEbits.STSEL = 0; // 1-stop bit
    U1MODEbits.PDSEL = 0; // No Parity, 8-data bits
    U1MODEbits.ABAUD = 0; // Auto-Baud Disabled
    U1MODEbits.BRGH = 0; // Low Speed mode
    //CALCULATE BAUD RATE
    //Fcy=Fosc/2 = 20 MHz
    //Baud Rate = Fcy/(16 x(UxBRG+1)) --> UxBRG=(Fcy/16 x Baud Rate)-1 --> UxBRG=129
    U1BRG = 259; // BAUD Rate Setting for 9600
    U1STAbits.UTXISEL0 = 0; // Interrupt after one Tx character is transmitted
    U1STAbits.UTXISEL1 = 0;
    IEC0bits.U1TXIE = 1; // Enable UART Tx interrupt
    U1STAbits.URXISEL = 0;//interrupt is generated each time a data word is transferred
                          //from the Receive Shift (UxRSR) register to the receive buffer
    IEC0bits.U1RXIE = 1; // Enable UART Rx interrupt
    IPC2bits.U1RXIP = 1; //Interrupcion de Rx de la UART tiene la prioridad mas baja
    U1MODEbits.UARTEN = 1; // Enable UART
    U1STAbits.UTXEN = 1; // Enable UART Tx
}
```

3.7 MÓDULO ECAN

El módulo ECAN (Enhanced Controller Area Network) es una interfaz serie, útil para comunicarse con otros módulos CAN o microprocesadores. Este interfaz/protocolo fue diseñado para hacer posible las comunicaciones en ambientes ruidosos. Los dispositivos de la familia dsPIC33FJXXXGPX06/X08/X10 poseen hasta un máximo de dos módulos ECAN. El módulo ECAN es un controlador de comunicaciones que implementa el protocolo CAN 2.0 A/B, tal como está definido en la especificación de Bosch. El módulo puede soportar las versiones del protocolo CAN 1.2, CAN 2.0A, CAN 2.0B Pasivo y CAN 2.0B Activo.

3.7.1 CARACTERÍSTICAS DEL MÓDULO ECAN

- Soporta tramas de datos estándar y extendidas.
- Longitud del campo de datos desde 0 hasta 8 bytes.
- Tasa binaria programable hasta 1Mb/s.
- Respuesta automática a peticiones de transmisión remota.
- Hasta 8 buffers de transmisión, cada uno con capacidad para 8 bytes de datos.
- Hasta 32 buffers de recepción, cada uno con capacidad para 8 bytes de datos.
- Hasta 16 filtros de aceptación de mensajes (identificador estándar/extendido).
- 3 máscaras de aceptación de filtros.
- Funcionalidad de arranque programable con filtro paso bajo integrado.
- Soporta direccionamiento DeviceNet.
- Modo Loopback programable.
- Señalización mediante interrupción de todos los estados de error de recepción y transmisión.
- Fuente de reloj programable.
- Enlace programable al módulo de captura para sincronización de la red.

El módulo ECAN consiste en un protocolo y un almacenamiento/control de mensajes. El protocolo se encarga de todas las funciones relacionadas con la transmisión y recepción de mensajes del bus CAN. Para transmitir un mensaje, primero hay que cargar los registros de datos apropiados. También hay registros específicos para

consultar el estado del bus y los posibles errores que se den en la comunicación. Cualquier mensaje que se detecte en el bus CAN, es inspeccionado en busca de errores y posteriormente comparado con los filtros de recepción para saber si dicho mensaje debe ser recibido y almacenado en los registros de recepción como se muestra en la figura 3.22.

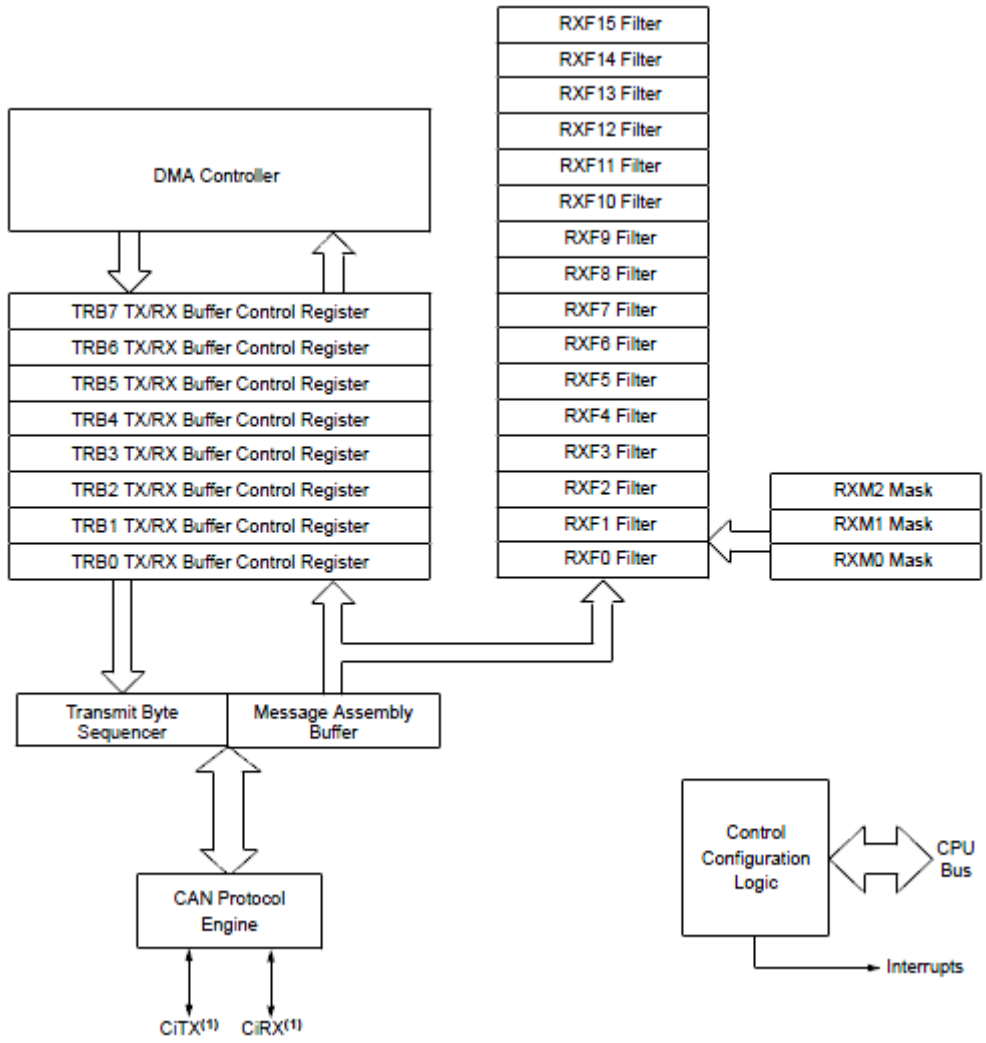


Figura 3.22 - Diagrama de bloques del módulo ECAN [5]

3.7.2 REGISTROS ECAN

El módulo ECAN tiene una gran cantidad de Registros de Función Especial (SFRs) que se usan para configurar los filtros de aceptación de los mensajes y los buffers. Para habilitar un uso eficiente del espacio de datos, hay varias series de SFRs mapeadas en el

mismo espacio de direcciones. El bit WIN del registro CiCTRL se usa para acceder a una de estas series de registros.

Si CiCTRL<WIN> = 1, los filtros de aceptación de mensajes y los registros que apuntan a los buffers de los filtros y máscaras son accesibles por la aplicación de usuario.

Si CiCTRL<WIN> = 0, los registros de control y estado del bus y los registros de transmisión y recepción de datos son accesibles por la aplicación de usuario.

Registros de control del Baud Rate:

- **CiCFG1 (registro de configuración del Baud Rate 1):** este registro contiene los bits de control que sirven para establecer el periodo de cada quantum de tiempo, utilizando el prescaler del baud rate, y especifica la anchura del salto de sincronización en unidades de quantum de tiempo.
- **CiCFG2 (registro de configuración del Baud Rate 2):** este registro se usa para programar el número de quantums de tiempo de cada segmento de bit CAN, incluyendo los segmentos de propagación y los segmentos de fase 1 y 2.

Registros de los filtros de los mensajes:

- **CiFEN1 (registro de habilitación del filtro de aceptación):** este registro habilita o deshabilita los filtros de aceptación del 0 al 15 para el filtrado de los mensajes.
- **CiRXFnSID (registro del identificador estándar del filtro de aceptación n (n=0-15)):** estos 16 registros especifican el identificador estándar del mensaje para los filtros de aceptación del 0 al 15. Los bits del identificador son comparados con los bits del mensaje entrante, para determinar si éste debe ser aceptado o rechazado. Estos registros solo son accesibles para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).
- **CiRXFnEID (registro del identificador extendido del filtro de aceptación n (n=0-15)):** estos 16 registros especifican el identificador extendido del mensaje para los filtros de aceptación del 0 al 15. Los bits del identificador son comparados con los bits del mensaje entrante para determinar si este debe ser aceptado o rechazado. Estos registros solo son accesibles para la aplicación de usuario cuando el bit WIN esta activo (CiCTRL1<0> = 1 = usar ventana de filtro).
- **CiRXMnSID (registro de la máscara para el identificador estándar del filtro de aceptación n (n=0-2)):** estos tres registros especifican los bits de la máscara para el identificador estándar de las Máscaras de Aceptación 0,1 y 2.

Cualquier filtro de aceptación puede seleccionar una de estas máscaras para comparar los bits del identificador selectivamente. Estos registros solo son accesibles para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).

- **CiRXMnEID (registro de la máscara para el identificador extendido del filtro de aceptación n (n=0-2)):** estos tres registros especifican los bits de la máscara para el identificador extendido de las Máscaras de Aceptación 0,1 y 2. Cualquier filtro de aceptación puede seleccionar una de estas máscaras para comparar los bits del identificador selectivamente. Estos registros solo son accesibles para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).
- **CiFMSKSEL1 (registro de selección de máscara para los filtros del 7 al 0):** este registro se usa para seleccionar la máscara de aceptación para los filtros del 0 al 7.
- **CiFMSKSEL2 (registro de selección de máscara para los filtros del 15 al 8):** este registro se usa para seleccionar la máscara de aceptación para los filtros del 8 al 15.
- **CiBUFPNT1 (registro puntero a buffer de los filtros 0 a 3):** este registro se usa para especificar el buffer que se utilizará para almacenar los mensajes aceptados por los filtros de aceptación 0 a 3. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).
- **CiBUFPNT2 (registro puntero a buffer de los filtros 4 a 7):** este registro se usa para especificar el buffer que se utilizará para almacenar los mensajes aceptados por los filtros de aceptación 4 a 7. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).
- **CiBUFPNT3 (registro puntero a buffer de los filtros 8 a 11):** este registro se usa para especificar el buffer que se utilizará para almacenar los mensajes aceptados por los filtros de aceptación 8 a 11. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar

  ventana de filtro).

- **CiBUFPNT4 (registro puntero a buffer de los filtros 12 a 15):** este registro se usa para especificar el buffer que se utilizará para almacenar los mensajes aceptados por los filtros de aceptación 12 a 15. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está activo (CiCTRL1<0> = 1 = usar ventana de filtro).

Registros de estado del buffer de los mensajes:

- **CiRXFUL1** (**registro 1 de buffer de recepción lleno):** junto con el registro CiRXFUL2, indica el estado de buffer lleno para los buffers de los mensajes de 0 a 31. Cuando un mensaje recibido se almacena en un buffer, se activa el respectivo flag de buffer lleno. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).
- **CiRXFUL2** (**registro 2 de buffer de recepción lleno):** junto con el registro CiRXFUL1, indica el estado de buffer lleno para los buffers de los mensajes de 0 a 31. Cuando un mensaje recibido se almacena en un buffer, se activa el respectivo flag de buffer lleno. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).
- **CiRXOVF1** (**registro 1 de buffer de recepción desbordado):** junto con el registro CiRXOVF2, indica el estado de desbordamiento para los buffers de los mensajes de 0 a 31. Cuando un mensaje recibido se almacena en un buffer y el flag de buffer lleno está activo, el mensaje se pierde y se activa el respectivo flag de desbordamiento. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).
- **CiRXOVF2** (**registro 1 de buffer de recepción desbordado):** junto con el registro CiRXOVF1, indica el estado de desbordamiento para los buffers de los mensajes de 0 a 31. Cuando un mensaje recibido se almacena en un buffer y el flag de buffer lleno está activo, el mensaje se pierde y se activa el respectivo flag de desbordamiento. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).

Registros de control/estado FIFO:

- **CiFCTRL (registro de control FIFO):** este registro controla el funcionamiento del buffer de recepción FIFO. Especifica la dirección de comienzo del buffer FIFO y el número de buffers reservados por el módulo ECAN en el espacio de direcciones DMA RAM. Este registro solo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).
- **CiFIFO (registro de estado FIFO):** este registro contiene punteros de lectura y escritura. El puntero de escritura apunta al buffer que contiene el dato que se ha recibido más recientemente. El puntero de lectura indica a la aplicación de usuario qué buffer será el siguiente en ser leído. Este registro sólo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).

Registros contadores de error y de control:

- **CiCTRL1 (registro de control 1):** este registro sirve para establecer los modos de funcionamiento del módulo ECAN.
- **CiCTRL2 (registro de control 2):** este registro contiene los bits de control del filtrado DeviceNet.
- **CiTRmnCON (registro de control del buffer TX/RX m (m=0,2,4,6; n=1,3,5,7)):** este registro configura y controla los buffers de los mensajes. Este registro sólo es accesible para la aplicación de usuario cuando el bit WIN está inactivo (CiCTRL1<0> = 0 = usar ventana de buffer).
- **CiEC (registro de cuenta de error transmisor/receptor):** este registro cuenta los errores de transmisión y recepción. La aplicación de usuario puede leer este registro para determinar el número actual de errores de transmisión y recepción.
- **CiRXD (registro receptor de datos):** este registro almacena temporalmente cada palabra recibida. Este es el registro desde el que el controlador DMA lee los datos en el buffer DMA.
- **CiTXD (registro transmisor de datos):** este registro almacena temporalmente cada transmisión. Este es el registro en el que el controlador DMA escribe los datos procedentes del buffer DMA.

3.7.3 BUFFERS ECAN

Los buffers para los mensajes ECAN están situados en el espacio de memoria DMA RAM. No hay SFRs (Registros de Función Especial) asociados al módulo ECAN. La aplicación de usuario puede escribir directamente en el área DMA RAM que esté configurada para actuar como buffers para los mensajes ECAN. La localización y el tamaño del área de buffer son definidos por la aplicación de usuario. En las figuras 3.23, 3.24, 3.25, 3.26, 3.27, 3.28, 3.29 y 3.30 se muestran los registros que almacenan los mensajes CAN.

Buffer 21-1: ECAN Message Buffer Word 0

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	SID10	SID9	SID8	SID7	SID6
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
SID5	SID4	SID3	SID2	SID1	SID0	SRR	IDE
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-13 **Unimplemented:** Read as '0'
bit 12-2 **SID<10:0>:** Standard Identifier bits
bit 1 **SRR:** Substitute Remote Request bit
When TXIDE = 0:
1 = Message will request remote transmission
0 = Normal message
When TXIDE = 1:
The SRR bit must be set to '1'
bit 0 **IDE:** Extended Identifier bit
1 = Message will transmit extended identifier
0 = Message will transmit standard identifier

Figura 3.23 - Registro que almacena la palabra 0 del mensaje CAN [5]

Buffer 21-2: ECAN Message Buffer Word 1

U-0	U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	—	EID17	EID16	EID15	EID14
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID13	EID12	EID11	EID10	EID9	EID8	EID7	EID6
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-12 **Unimplemented:** Read as '0'
bit 11-0 **EID<17:6>:** Extended Identifier bits

Figura 3.24 - Registro que almacena la palabra 1 del mensaje CAN [5]

Buffer 21-3: ECAN™ Message Buffer Word 2

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EID5	EID4	EID3	EID2	EID1	EID0	RTR	RB1
bit 15							bit 8

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	RB0	DLC3	DLC2	DLC1	DLC0
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-10 **EID<5:0>:** Extended Identifier bits
bit 9 **RTR:** Remote Transmission Request bit
When TXIDE = 1:
1 = Message will request remote transmission
0 = Normal message
When TXIDE = 0:
The RTR bit is ignored.
bit 8 **RB1:** Reserved Bit 1
User must set this bit to '0' per CAN protocol.
bit 7-5 **Unimplemented:** Read as '0'
bit 4 **RB0:** Reserved Bit 0
User must set this bit to '0' per CAN protocol.
bit 3-0 **DLC<3:0>:** Data Length Code bits

Figura 3.25 - Registro que almacena la palabra 2 del mensaje CAN [5]

Buffer 21-4: ECAN™ Message Buffer Word 3

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 1							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 0							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-8 ECAN Message byte 1
bit 7-0 ECAN Message byte 0

Figura 3.26 - Registro que almacena la palabra 3 del mensaje CAN [5]

Buffer 21-5: ECAN™ Message Buffer Word 4

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 3							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 2							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-8 ECAN Message byte 3
bit 7-0 ECAN Message byte 2

Figura 3.27 - Registro que almacena la palabra 4 del mensaje CAN [5]

Buffer 21-6: ECAN™ Message Buffer Word 5

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 5							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 4							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-8 ECAN Message byte 5
bit 7-0 ECAN Message byte 4

Figura 3.28 - Registro que almacena la palabra 5 del mensaje CAN [5]

Buffer 21-7: ECAN™ Message Buffer Word 6

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 7							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
Byte 6							
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-8 ECAN Message byte 7
bit 7-0 ECAN Message byte 6

Figura 3.29 - Registro que almacena la palabra 6 del mensaje CAN [5]

Buffer 21-8: ECAN™ Message Buffer Word 7

U-0	U-0	U-0	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	FILHIT4	FILHIT3	FILHIT2	FILHIT1	FILHIT0
bit 15							bit 8

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-13 **Unimplemented:** Read as '0'

bit 12-8 **FILHIT<4:0>:** Filter Hit Code bits
Encodes number of filter that resulted in writing this buffer (only written by module for receive buffers, unused for transmit buffers).

bit 7-0 **Unimplemented:** Read as '0'

Figura 3.30 - Registro que almacena la palabra 7 del mensaje CAN [5]

3.7.4 MODOS DE FUNCIONAMIENTO

El módulo ECAN puede funcionar en seis modos diferentes, seleccionables por el usuario: modo de configuración, modo normal de funcionamiento, modo de solo escucha, modo de escucha de todos los mensajes, modo de bucle cerrado y modo deshabilitar.

Modo de configuración:

Después de un reset hardware, el módulo ECAN se encuentra en el modo de configuración (bits OPMODE<2:0>=100). Los contadores de error están a cero y todos

los registros contienen sus valores de reset. Para poder modificar los registros de control del tiempo de bit del módulo ECAN (CiFG1 y CiFG2), el módulo debe estar en el modo de configuración.

Modo normal de funcionamiento:

En el modo normal de funcionamiento, el módulo ECAN puede transmitir y recibir mensajes CAN. El modo normal de funcionamiento se solicita después de una inicialización programando los bits REQOP<2:0> del registro (CiCTRL<10:8>) a 000. Cuando los bits OPMODE<2:0> pasan a valer 000, el módulo ECAN se encuentra en el modo normal de funcionamiento.

Modo de sólo escucha:

Este modo de funcionamiento se usa principalmente para la monitorización del bus, sin participación real en el proceso de transmisión. El nodo que esté en el modo de sólo escucha no genera tramas de error ni de confirmación.

Modo de escucha de todos los mensajes:

El modo de escucha de todos los mensajes se usa para depurar el sistema. Su funcionamiento consiste en que todos los mensajes son recibidos, independientemente de su identificador e incluso cuando hay un error. Si el módulo ECAN está funcionando en el modo de escucha de todos los mensajes, la transmisión y la recepción funcionan igual que en el modo normal de funcionamiento, excepto porque los mensajes recibidos con un error también se almacenan en el buffer de recepción.

Modo de bucle cerrado:

El modo de bucle cerrado se usa para que el módulo pueda recibir su propio mensaje y así probarse a sí mismo. En este modo, la ruta de transmisión y la de recepción están conectadas internamente. Se proporciona un “simulacro” de confirmación, eliminando por tanto la necesidad de otro nodo que proporcione el bit de confirmación.

Modo deshabilitar:

El modo deshabilitar se usa para asegurar un apagado seguro antes de poner el microprocesador en modo Sleep o Idle. El controlador ECAN espera hasta que detecta un secuencia para poner al bus en estado idle (11 bits a nivel alto) antes de cambiar de modo. Cuando el módulo está en el modo deshabilitar, detiene sus fuentes de reloj, pero esto no tiene efecto en la CPU ni en otros módulos. El módulo ECAN se “despierta” cuando se produce actividad en el bus o cuando la CPU pone los bits OPMODE<2:0> a 000. El pin CiTX permanece en estado recesivo mientras el módulo esté en este modo de funcionamiento.

3.8 CONFIGURACIÓN DEL CONTROLADOR DMA

El módulo ECAN utiliza una porción del espacio de memoria DMA RAM como buffer para almacenar los mensajes procedentes del bus CAN y permitir así la transmisión y recepción de los mismos. El número de buffers a utilizar se especifica con los bits DMABS<2:0> (DMA Buffer Size) del registro CiFCTRL (ECAN FIFO Control Register). El registro DMAxSTA marca el comienzo del área de buffer CAN. El

controlador DMA transporta datos entre el módulo ECAN y el DMA sin la intervención de la CPU.

También proporciona ocho canales para la transferencia de datos entre la memoria DMA RAM y los periféricos del microprocesador. Para soportar la transmisión y la recepción de los mensajes CAN, se necesitan dos canales DMA. Cada canal tiene asociado un registro de petición (DMAxREQ) para asignar un evento basándose en qué tipo de transferencia ocurre, tal y como se muestra en la siguiente tabla:

Tabla 16 – Petición del canal DMA [2]

Event	IRQ Number to Initialize DMAxREQ Register
ECAN1 Reception	34
ECAN1 Transmission	70
ECAN2 Reception	55
ECAN2 Transmission	71

En la figura 3.32 se muestra el espacio de memoria DMA RAM

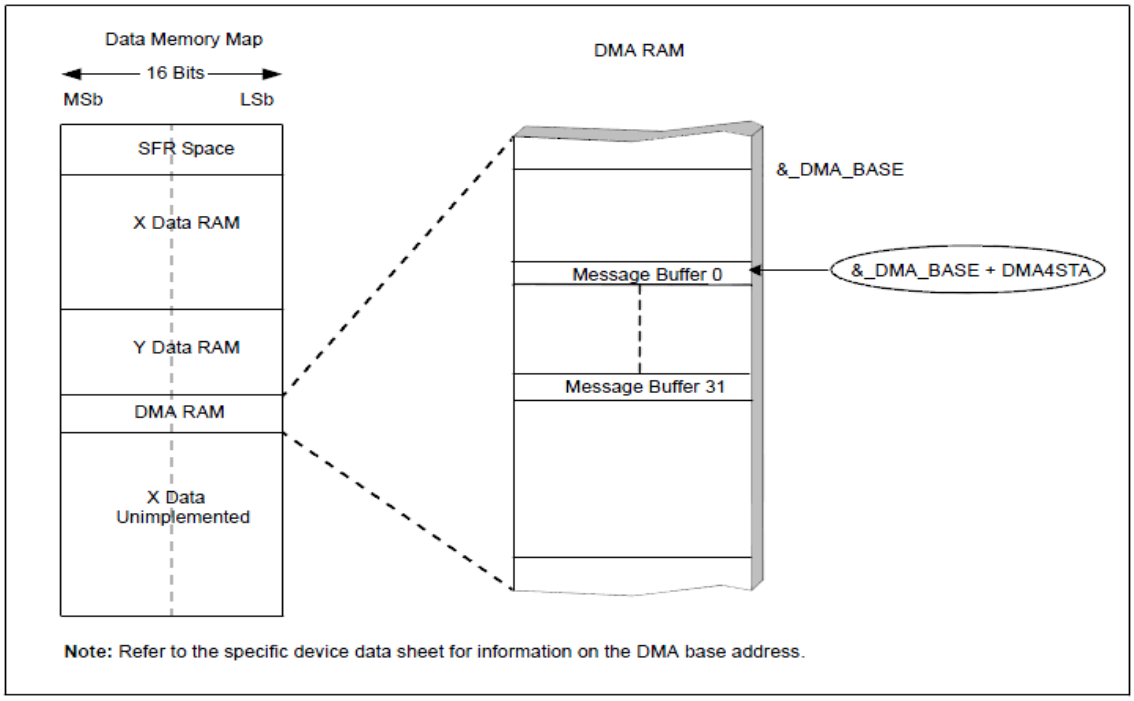


Figura 3.32 – Utilización de la memoria como buffer para los mensajes ECAN [2]

3.8.1 FUNCIONAMIENTO DMA PARA TRANSMISIÓN DE DATOS

La aplicación de usuario selecciona un mensaje para ser transmitido activando el bit de petición de envío de mensaje (TXREQ) en el registro CiTRmnCON. El controlador ECAN utiliza el DMA para leer el mensaje del buffer y transmitir el mensaje al bus. El módulo ECAN genera una interrupción de transmisión de datos para empezar un ciclo DMA. En respuesta a la interrupción, el canal DMA que está configurado para transmitir mensajes, lee el mensaje del buffer de la DMA RAM y almacena el mensaje en el registro de transmisión de datos ECAN (CiTXD). Por cada mensaje transmitido por el controlador ECAN, se transfieren 8 palabras.

3.8.2 FUNCIONAMIENTO DMA PARA RECEPCIÓN DE DATOS

Cuando el controlador ECAN completa la recepción de un mensaje (8 palabras), el mensaje se transfiere al buffer en la DMA RAM mediante el controlador DMA. El módulo ECAN genera una interrupción de recepción de datos para empezar un ciclo DMA. En respuesta a la interrupción, el canal DMA que está configurado para recibir mensajes, lee el mensaje del registro de recepción de datos ECAN (CiRXD) y almacena el mensaje en el buffer DMA RAM. Se transfieren 8 palabras por cada mensaje recibido por el controlador ECAN.

3.9 APLICACIÓN DEL EMISOR

La aplicación que se ejecuta en el circuito emisor, tiene como misión recibir las tramas que llegan a través del bus CAN procedentes del vehículo y transmitir los datos que llegan en esas tramas a través de la UART del microprocesador al chip de radio. Para ello se usa un autómata con tres estados, que son los que se muestran en la figura 3.33:

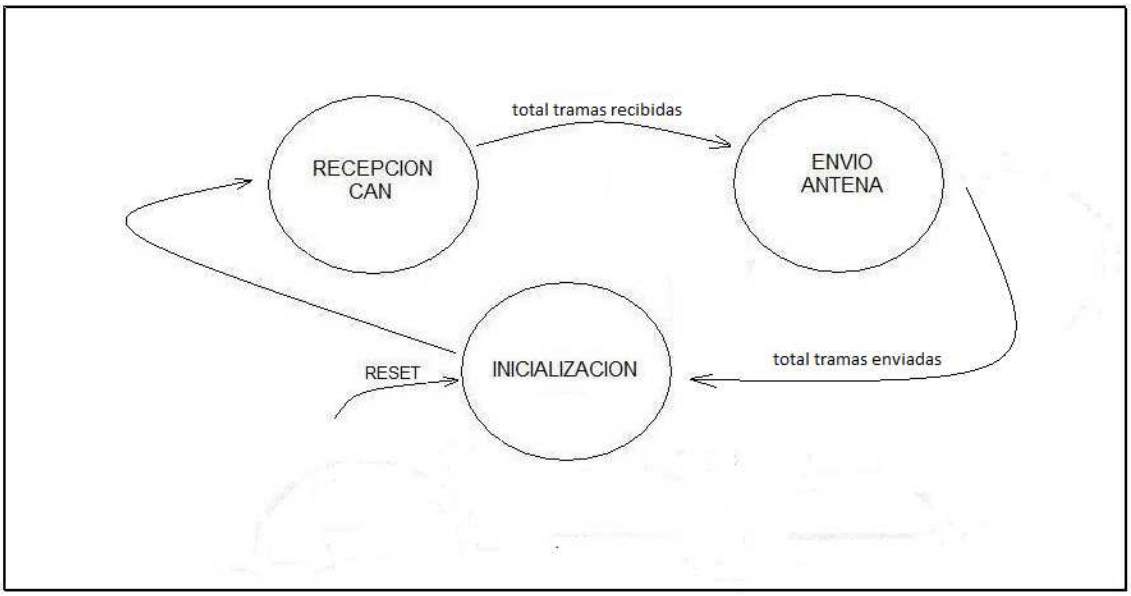


Figura 3.33 – Diagrama de estados de la aplicación emisora

Tras la inicialización de los recursos del microprocesador que se van a usar en la aplicación, como son el oscilador interno, la UART1, el DMA y los pines de entrada/salida, la aplicación entra en un bucle infinito en el que se ejecuta el autómata anteriormente citado. El primer estado por el que pasa el autómata es el de inicialización.

INICIALIZACIÓN:

Este estado sirve para resetear todas las variables que se han usado a lo largo del programa, de manera que la aplicación quede preparada para volver a recibir y enviar datos. Tras esta inicialización de las variables se transita automáticamente al estado recepción can, sin necesidad de que ocurra un evento concreto que produzca esta transición.

RECEPCIÓN CAN:

En este estado se van habilitando uno a uno los filtros de aceptación de los mensajes que se quieren recibir. Esto debe hacerse así ya que debido al gran flujo de datos que circula por el bus CAN, si se configuran varios filtros de aceptación al mismo tiempo, se activan varios bits que indican la recepción de un mensaje, de manera que es imposible discriminar por qué mensaje se ha producido la interrupción. Una vez que se produce el evento de la recepción de la trama, se ejecuta la interrupción del módulo ECAN, en la cual se almacenan los datos en el buffer de recepción del DMA y se activa el flag que indica que se ha recibido una trama procedente del bus CAN. Este proceso se repite hasta que se han almacenado todas las tramas deseadas. Una vez almacenadas las tramas se transita al estado envío antena.

ENVÍO ANTENA:

En este estado se construye la trama, añadiéndole a los datos la información redundante necesaria (el identificador de la trama para que el receptor sepa qué dato le está llegando y un carácter para indicarle al receptor el fin de trama). Una vez construida la trama, ésta es enviada a través de la UART. Una vez que se han enviado todas las tramas, se vuelve al estado de inicialización.

3.10 APLICACIÓN DEL RECEPTOR CON LABVIEW

Los datos son recogidos en un punto remoto por el receptor de radio (el cual está pinchado en la placa de desarrollo del kit de desarrollo Xbee Pro 868) y son enviados por USB a un ordenador donde los datos son monitorizados en una aplicación de LabVIEW.

Esta aplicación va mostrando los datos a medida que van llegando por el puerto USB y almacena dichos datos en arrays hasta que, transcurrido un cierto tiempo, los datos son guardados en un fichero para permitir un posterior estudio. Este proceso se repite cíclicamente de manera que se recogen datos durante treinta segundos, se escriben en el fichero y se vuelve al estado donde se leen datos nuevamente. Así hasta que el usuario decida finalizar la aplicación.

La aplicación está estructurada por tanto y al igual que la aplicación emisora como una máquina de estados, en este caso con dos únicos estados; uno llamado MEDIR en el que se recogen los datos recibidos por el puerto USB y se muestran en pantalla y otro llamado ESCRIBE FICHERO, en el cual se realizan las escrituras en el fichero de los datos que llegaron durante los últimos treinta segundos.

A continuación, en las imágenes 3.34 y 3.35 se puede ver el aspecto del Panel Frontal de la aplicación. Dicho panel frontal está compuesto por dos pestañas; una de ellas donde se visualizan todos los datos de interés que indican el comportamiento del coche (figura 3.34) y otra donde se muestran diferentes variables de error y de estado (figura 3.35).

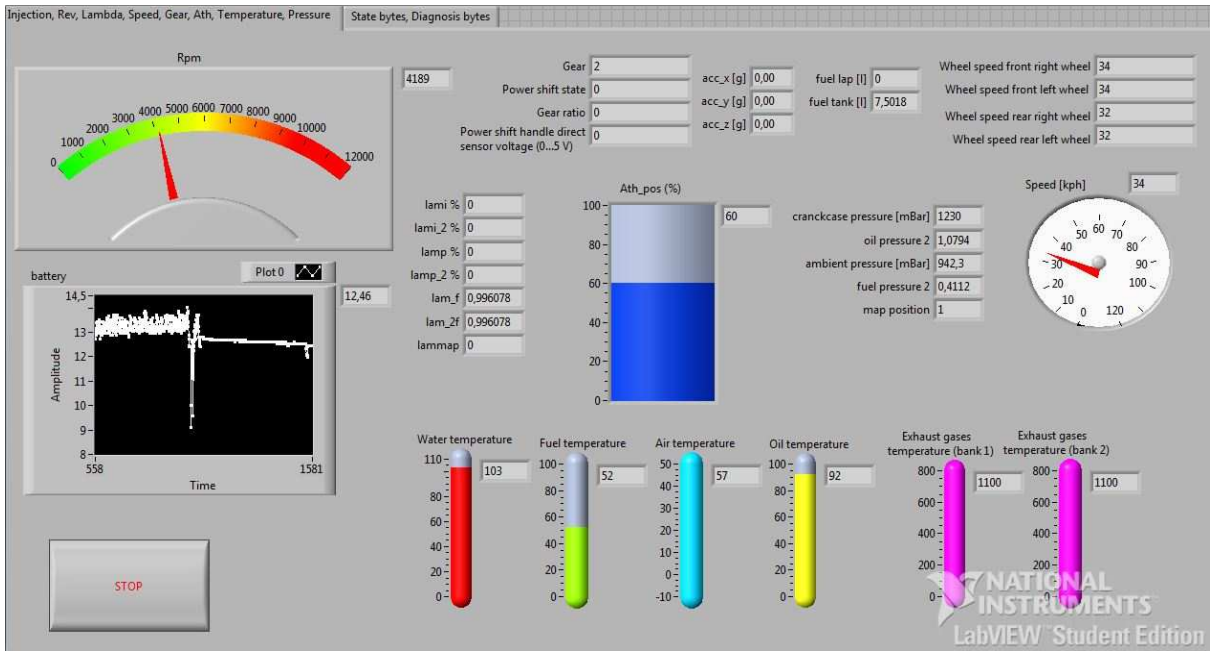


Figura 3.34 – Panel Frontal de la aplicación LabVIEW (datos del vehículo)

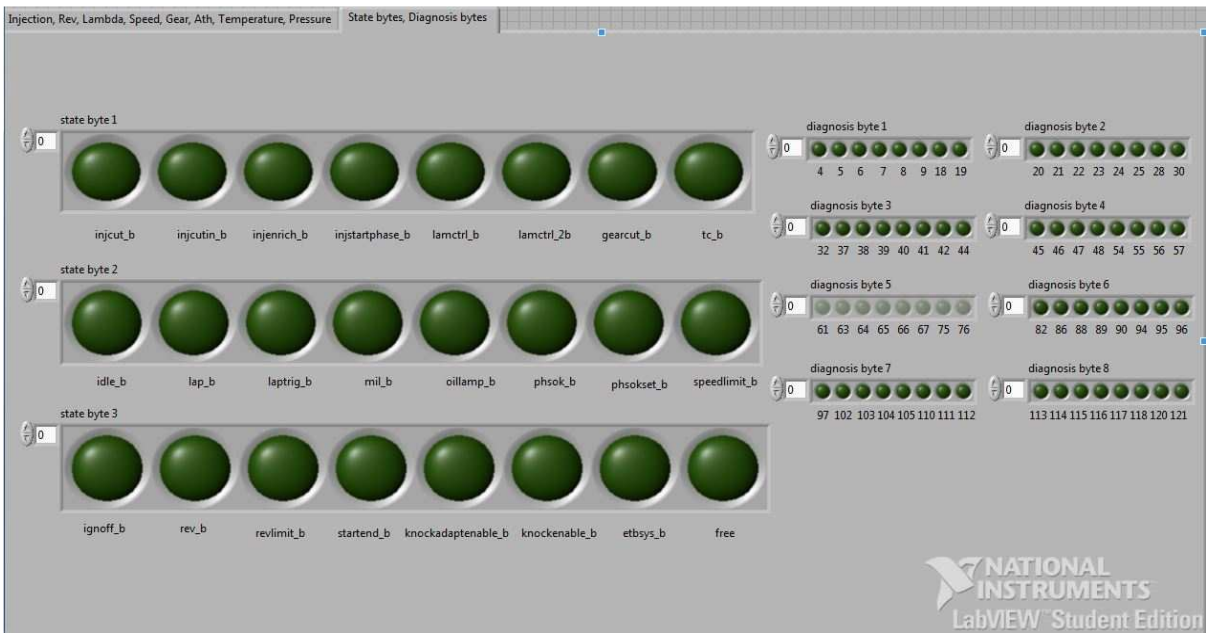


Figura 3.35 – Panel Frontal de la aplicación LabVIEW (bytes de estado y diagnóstico)

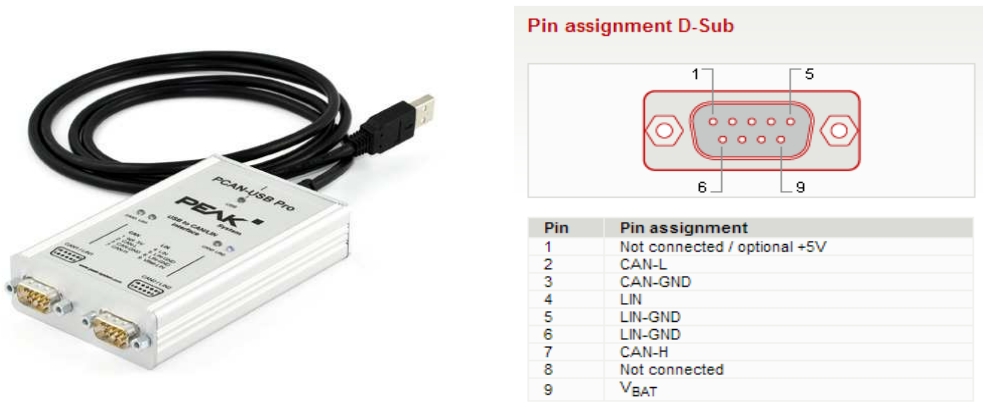
CAPÍTULO 4: PRUEBAS Y RESULTADOS

Para realizar pruebas al sistema se necesitaba algún sistema que simulara el funcionamiento del bus CAN y que proporcionara los datos de entrada al sistema. Era posible conectarse a la centralita del vehículo, y de hecho se ha hecho, pero al no estar el motor listo todavía en el momento de las pruebas y al no estar por tanto la centralita conectada a ningún sensor, la mayoría de los campos de datos estaban vacíos, por lo que parecía más interesante realizar pruebas con un sistema que permitiera simular el comportamiento real de la centralita cuando tuviera conectados los sensores del motor aportando datos.

Por suerte en el INSIA se contaba con una interfaz que permitía justamente realizar estas pruebas.

4.1 SIMULADOR DE BUS CAN PCAN-USB PRO

El adaptador PCAN-USB Pro es una interfaz que permite la conexión entre un PC y redes LIN o CAN. Permite la conexión de dos buses al mismo tiempo. Se conecta al PC a través de USB y a la red CAN/LIN mediante conector serie DB9 con la distribución de pines que muestra la figura 4.1.



Pin	Pin assignment
1	Not connected / optional +5V
2	CAN-L
3	CAN-GND
4	LIN
5	LIN-GND
6	LIN-GND
7	CAN-H
8	Not connected
9	V_{BAT}

Figura 4.1 – Interfaz PCAN-Usb Pro

4.1.1 FUNCIONAMIENTO DEL SOFTWARE PCAN VIEW

Es un software gratuito que se puede descargar de la página web de PEAK Systems (http://www.peak-system.com/PCAN-View.242.0.html?&L=1) que permite monitorizar, enviar y grabar el tráfico de datos CAN. Permite enviar mensajes tanto manualmente como periódicamente a una velocidad configurable de hasta 1Mb/s.

Combinando estos dos recursos se han generado las tramas de datos que simulaban el funcionamiento real de la centralita.

4.2 PRUEBAS EN EL LABORATORIO

Esta prueba puso de manifiesto lo citado en el apartado 3.9 y es que si se habilita la recepción de varios mensajes CAN a la vez, se activan varios bits del registro C1RXFUL1 que indican qué mensaje ha provocado la interrupción, con lo cual se deben ir habilitando uno a uno la recepción de los mensajes ya que si no es imposible averiguar en la interrupción del bus CAN qué mensaje es el que se debe almacenar en ese momento. A continuación se muestran varias capturas de pantalla sobre las pruebas realizadas.

En la primera prueba se configura el envío periódico de dos tramas (identificadores 0x770 y 0x771) en el software PCAN View y se trabaja con el microprocesador en el modo de depuración. Se pone un punto de ruptura en la interrupción del bus CAN y se comprueba el valor de los registros C1RXFUL1 (indica qué buffer de recepción se ha llenado) y C1RXOVF1 (indica en qué buffers se ha producido desbordamiento por la llegada de un mensaje sin haber sido leído el anterior).

Se lanza el programa y se observa que la primera vez que se entra en la interrupción, el valor de los registros es C1RXFUL1 = 0x0003 y C1RXOVF1 = 0x0003, lo cual indica que tanto en el buffer de recepción del mensaje con identificador 0x770 como en el del mensaje con identificador 0x773 ha habido desbordamiento.

En la figura 4.2 se muestra una captura de pantalla del software PCAN-View donde se muestra el resultado de la prueba.

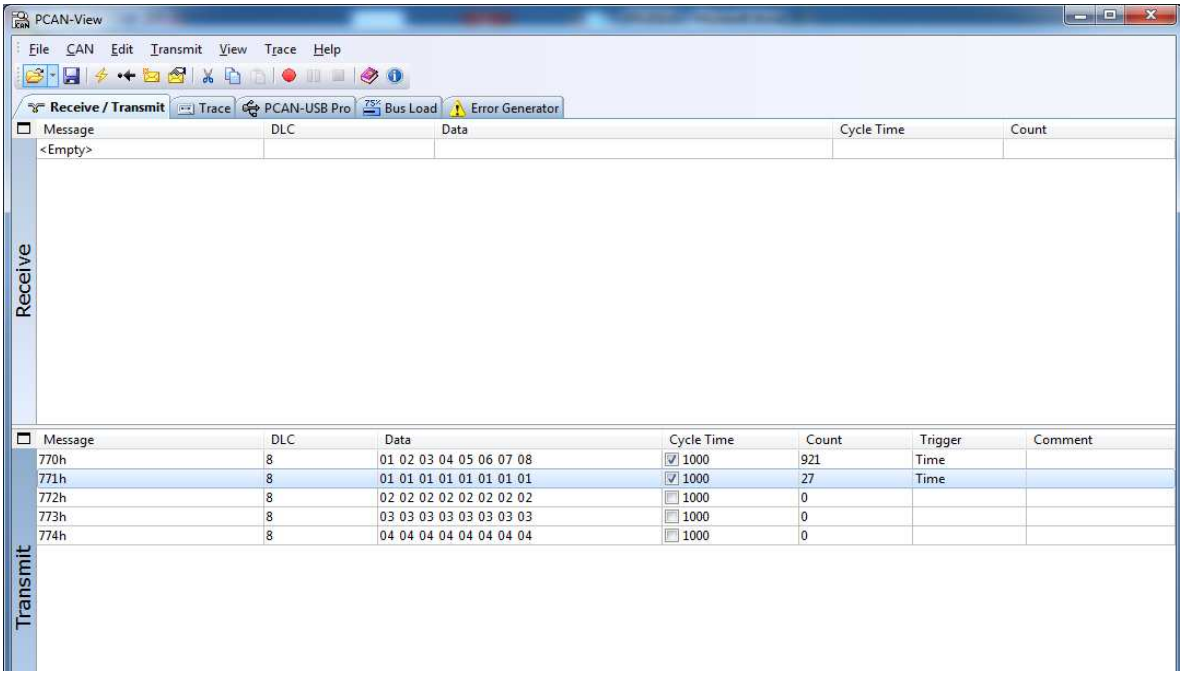


Figura 4.2 – PCAN- View, transmisión de dos mensajes

Y en la figura 4.3 se muestra una captura de pantalla del software MPLAB donde aparece resaltado el estado de los registros de interés en esta prueba.

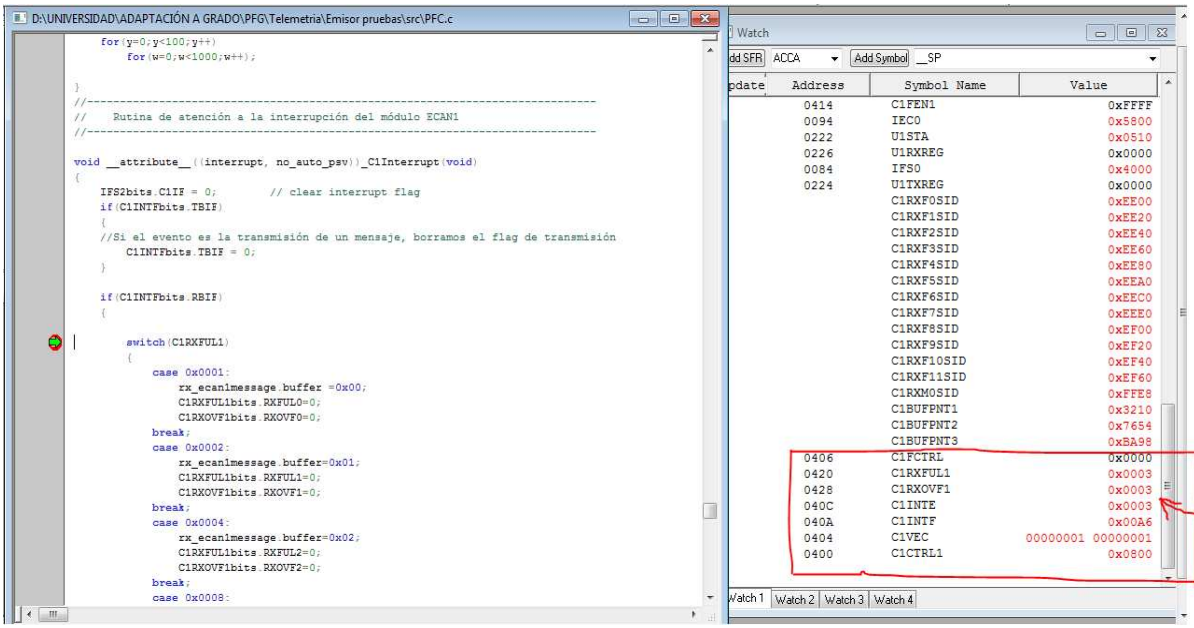


Figura 4.3 – Registros C1RXFUL1 y C1RXOVFL, transmisión de dos mensajes

En la siguiente prueba se envía solamente un mensaje (identificador 0x770) también periódicamente. Al entrar en la interrupción se observa que el valor de los registros es C1RXFUL1 = 0x0001 y C1RXOVF1 = 0x0001.

De esta manera ya se puede saber que buffer es el que se ha llenado y por tanto cual es el que hay que leer. Sin embargo sigue produciéndose desbordamiento ya que la velocidad de envío del mensaje (y este es el funcionamiento real de la centralita) es mayor que la velocidad a la que se puede ejecutar el código. Este problema se soluciona deshabilitando la recepción de ese mensaje al final de la interrupción y borrando el bit de desbordamiento, de manera que no se vuelva a producir una interrupción por el mismo mensaje.

En la figura 4.4 se muestra la captura de pantalla del software PCAN View correspondiente a esta prueba.

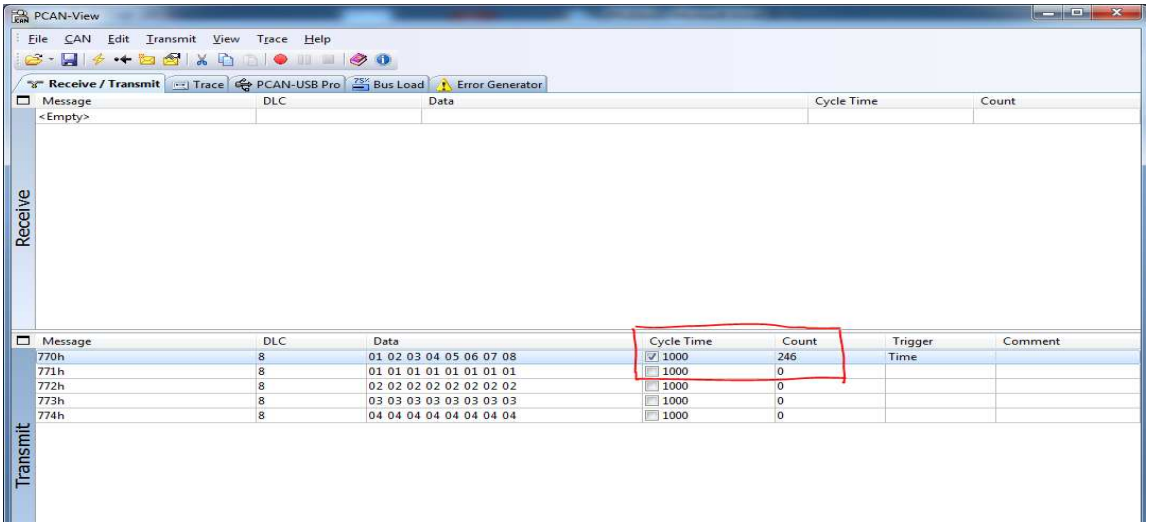


Figura 4.4 – PCAN-View, transmisión de un mensaje

Y en la figura 4.5 se muestra nuevamente una captura de pantalla de MPLAB donde aprece resaltado el estado de los registros C1RXFUL1 y C1RXOVFL1

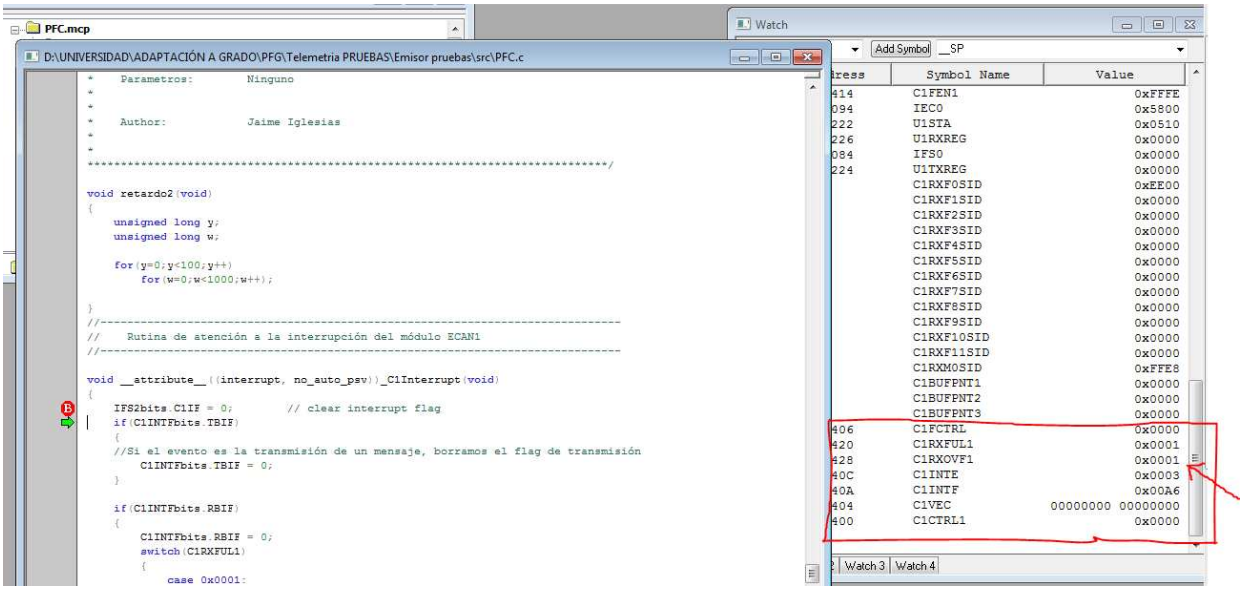


Figura 4.5 – Registros C1RXFUL1 y C1RXOVFL, transmisión de un mensaje

En cuanto a las pruebas de radio, se hicieron varios test de alcance con resultados muy satisfactorios. Se logró comunicar perfectamente entre dos habitaciones situadas en dos edificios diferentes, separados unos cincuenta metros.

Sin embargo, cabe la duda de cómo se comportará el sistema una vez esté montado en el coche y funcione en movimiento. Lamentablemente, en el momento de la redacción de este documento el coche no estaba listo para rodar y estas pruebas no se han podido realizar.

Una vez solucionados los problemas de recepción de datos del bus CAN y una vez se tenían realizadas las PCBs, se hicieron pruebas de funcionamiento real del sistema, con las PCBs conectadas a la centralita MS3 y el receptor conectado al ordenador donde se monitorizaban los datos en LabVIEW.

Estas pruebas sirvieron para depurar pequeños errores de programación en LabVIEW por los que no se visualizaban correctamente algunas variables.

Continuando con la realización de las pruebas, se puso de manifiesto que la recepción de datos se detenía más o menos a los quince minutos de haber puesto el sistema en funcionamiento. Al comprobar que este comportamiento se producía siempre y tras cerciorarse de que no era problema del programa que ejecutaba el microprocesador, se realizó una prueba con las placas de desarrollo para ver si también se detenía la recepción de datos. Se comprobó que efectivamente la recepción de datos también se interrumpía, pero transcurrido un tiempo diferente.

Esta prueba ponía de manifiesto que el problema estaba en los módulos de radio. Para solucionar esta anomalía se contactó con el fabricante y este indicó que la detención de la transmisión de datos se debía a una característica de los módulos de radio denominada ciclo de trabajo.

El ciclo de trabajo del chip de radio es del 10% de una hora. Esto significa que se pueden transmitir datos de manera continua durante seis minutos, es decir, que para poder transmitir sin que el sistema se detenga es necesario espaciar las transmisiones. Esto es un hándicap importante para la aplicación que se ha desarrollado, ya que si se espacian las transmisiones de datos, la frecuencia de refresco de los mismos sería demasiado baja.

Como solución a este problema se propone rediseñar la placa de circuito impreso incluyendo un circuito que provoque un reset en la alimentación del transmisor de radio, de manera que la cuenta del tiempo del ciclo de trabajo comenzaría de nuevo y el sistema no estaría parado hasta que se cumpliera una hora desde el momento en que se encendió el sistema. Por falta de tiempo esta nueva PCB no se ha podido probar, por lo que entra dentro de las tareas futuras y mejoras que se podrían hacer al sistema.

4.3 PRUEBAS DE CAMPO

Una vez se tenía depurado el sistema había que montarlo en el coche para poder realizar pruebas de funcionamiento en condiciones reales. Tras pensar en varias posibilidades en cuanto a dónde colocarlo, si encapsular las dos PCBs en una caja o colocarlas por separado o incluso en situar la antena en el morro del coche y conectarla a las PCBs mediante un cable coaxial en lugar de enroscar la antena directamente en el módulo de radio.

Finalmente se optó por colocar las dos PCBs juntas en una caja y conectar la antena directamente al chip de radio. En la figura 4.6 se puede ver todo el montaje finalizado. Los motivos de optar por esta solución fueron varios. Por un lado el hacer un sistema compacto al tener las dos PCBs en la misma caja. Por otro lado, el hecho de poner esa caja encima del motor se debe a la cercanía con la centralita de modo que los cables que había que tirar hasta la caja de telemetría resultarían mucho más cortos que si se colocaba la caja en la parte delantera del coche. Y por último se desechó la opción de colocar la antena en el morro y conectarla mediante cable al chip de radio porque no se tenía el cable, con lo cual hubiera supuesto un gasto más, el cable podía introducir pérdidas que afectaran al funcionamiento del sistema y no se tenía mucho tiempo para probar alternativas de las que se tenía dudas de su éxito.

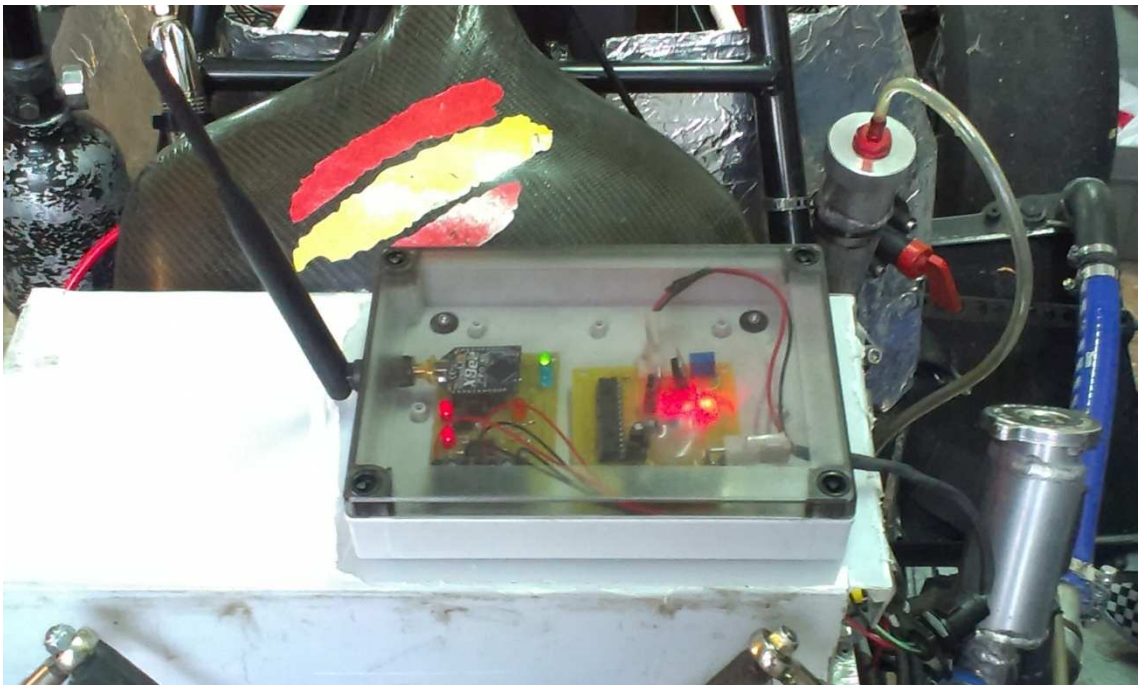


Figura 4.6 – Caja de Telemetría montada en el coche

Una vez montado el sistema se realizaron pruebas en pista detectando algunos problemas.

Se observó que muchas tramas se recibían erróneamente en la aplicación de LabVIEW, representándose en pantalla datos erróneos.

Tras reflexionar sobre cuál podía ser la causa de que el sistema estuviera fallando cuando en las anteriores pruebas había funcionado correctamente, se pensó en dos causas posibles. La primera, que las vibraciones del motor estuvieran afectando a la señal de entrada al chip de radio ya que todas las demás pruebas se habían efectuado con el motor apagado y las vibraciones que produce el motor son muy fuertes y la segunda que el transmisor estuviera emitiendo con demasiada potencia y esto estuviera saturando el receptor.

Para tratar de reducir las vibraciones que sufrían las PCBs se forró la base de la caja de telemetría con papel burbuja, no obteniéndose una mejoría clara, por lo que se optó por reprogramar los chips de radio para que trabajaran a menos potencia, consiguiendo de este modo solucionar el problema.

CAPÍTULO 5: CONCLUSIONES

En este capítulo se abordarán las conclusiones de este trabajo, tras el estudio y análisis de los resultados obtenidos después de realizar las diferentes pruebas del sistema. Asimismo, se compararán los resultados de este proyecto con un proyecto anterior en el que se utilizaron otros módulos de radio, las PCBs fueron fabricadas con otro método y no se utilizaba LabVIEW para monitorizar los datos.

Las conclusiones que se extraen del proyecto a nivel general son my positivas, ya que se cree que se ha mejorado en varios aspectos lo que había, dejando al UPM Racing un sistema funcional que facilitará el control de los vehículos durante las sesiones de prueba ya que se tendrán datos en tiempo real del comportamiento del coche y no habrá que detener el coche para descargar los datos del sistema de adquisición como ocurría hasta ahora.

El objetivo de este Proyecto Fin de Grado era mejorar un proyecto ya existente fundamentalmente en tres aspectos:

- El primero era conseguir que el sistema fuera capaz de transmitir el flujo de datos que proporcionaba la centralita del vehículo ya que el sistema anterior solo era capaz de transmitir mensajes CAN de uno en uno. Esto se debía a un fallo de programación que, como se explica en el apartado 3.9 de este documento, provocaba que no se pudiera saber que mensaje había llegado en cada momento y que se produjeran interrupciones continuas en el microprocesador de manera que el programa se quedaba bloqueado en uno de los estados del diagrama de estados.
  Una vez corregido este error el programa era capaz de transmitir los datos proporcionados por la centralita del coche, con lo que el primer objetivo quedaba cumplido. El único aspecto negativo a comentar es que como la recepción de los mensajes se realiza de uno en uno hay mensajes que se pierden, pero esto no es significativo ya que al tener una limitación en la tasa de datos RF de 24kbps y enviar la centralita a una velocidad de 1Mbps, la pérdida de datos era inevitable.

- Otro objetivo fundamental era mejorar el alcance del sistema que con los módulos de radio utilizados hasta el momento era teóricamente de 300 metros, pero en la práctica de tan sólo 60 metros, insuficiente para poder recibir datos desde cualquier punto del circuito. Para conseguir este objetivo se sustituyeron los chips que había por el kit de desarrollo Xbee Pro 868, cuyos módulos de radio presentan un alcance teórico de 500 metros bajo techo y de hasta 40km en espacios al aire libre y visión directa. Con el empleo de estos módulos de radio se solucionó completamente cualquier problema de alcance. Incluso hay que

trabajar a niveles de potencia bajos para no saturar al receptor cuando el coche pasa por un punto del circuito cercano a donde se coloca el receptor.

- Y el último objetivo que se había marcado al comienzo del proyecto era mejorar la interfaz en la que se visualizaban los datos. Para ello se eligió LabVIEW por ser un lenguaje conocido, especialmente pensado para este tipo de aplicaciones donde se monitorizan datos y con el que se pueden desarrollar interfaces muy atractivas visualmente.

Además se han adquirido nuevos conocimientos, como el manejo del nuevo software de diseño de circuitos impresos creado por RS, llamado DesignSparkPCB y se han afianzado otros como la programación en LabVIEW o el proceso de fabricación de los circuitos impresos.

A continuación se hace una comparativa entre el proyecto desarrollado y un proyecto anterior en tres aspectos fundamentalmente: los módulos de radio empleados, el método de fabricación de los circuitos impresos y la manera de visualizar los datos.

5.1 Xbee Pro 868 vs Linx RF Modules

En el anterior proyecto se utilizaron unos módulos de radio de la marca Linx Technologies. El alcance de dichos módulos era tan solo de 300 metros (dato del fabricante) y en funcionamiento real el alcance obtenido era de sólo 75 metros, que era muy poco para la aplicación desarrollada.

Otro inconveniente grande en el proyecto anterior fue que no se tenían como en este placas de desarrollo para los módulos de radio. Además dichos módulos de radio tenían encapsulado SMD y eran muy frágiles de manera que al desoldarlos era muy difícil que no se estropearan. Los módulos usados en este proyecto tienen soldados dos tiras de pines de manera que se pinchan en un zócalo con lo que es muy fácil moverlos de la placa de pruebas a la placa de tu prototipo.

Otra ventaja de los nuevos módulos de radio es que ellos solos implementan un control de errores enviando el receptor confirmación al emisor de que ha recibido la trama o en caso de no haberla recibido solicita la retransmisión de la misma. Los módulos de Linx Technologies eran más sencillos y era el usuario quién debía implementar el protocolo de comunicaciones desarrollando el código por si mismo.

Sin embargo, los módulos de radio Xbee Pro 868 presentan una desventaja importante y es la del ciclo de trabajo, que sólo permite transmitir 6 minutos por hora, mientras que los módulos de Linx Technologies no presentaban ninguna limitación en este aspecto.

5.2 PCBs: Ácidos e Insoladora vs Pasta conductora

Otro cambio respecto al proyecto anterior ha sido el método de fabricación de PCBs. En el pasado proyecto se utilizó un método novedoso de fabricación de circuitos impresos que consistía en metalizar las vías mediante la aplicación de una pasta conductora. Este método daba muchos problemas. Primero se colocaba un plástico en la placa y posteriormente se realizaban los taladros que posteriormente se metalizaban. El problema era que en los taladros quedaban rebabas que provocaban que al aplicar la pasta conductora esta se esparciera por la cara de abajo de la placa por fuera de los pads, provocando cortocircuitos que posteriormente y tras curar la pasta en un horno para que endureciera, había que corregir rascando para eliminar los restos de pasta conductora. Además, era habitual que la pasta no cubriese perfectamente alguna vía con lo cual ese camino de la placa quedaba sin continuidad, con lo que al final el tiempo que se trataba de ahorrar respecto al método de los ácidos en el que tras el taladrado había que soldar una a una las vías por ambas caras de la placa para provocar que hubiera continuidad entre ambas caras no era tal, ya que se perdía bastante tiempo en comprobar que todos los taladros habían quedado metalizados correctamente y en corregir los cortocircuitos que hubiera provocado la pasta conductora.

Volviendo al método tradicional de insolación y atacado químico se ha ganado en tiempo y fiabilidad.

5.3 Hyperterminal vs LabVIEW

Por, último la otra gran novedad incorporada en este proyecto es el método de visualización de los datos. En el proyecto anterior se utilizaba el programa de Windows Hyperterminal mientras que en el proyecto actual sea utilizado LabVIEW.

Mientras que Hyperterminal solamente permite imprimir caracteres ASCII, LabVIEW es un entorno de programación visual que cumple mucho mejor con las necesidades de este proyecto ya que proporciona un entorno de visualización de los datos mucho más atractivo y además es más sencillo de programar que el código que había que desarrollar en lenguaje C para que los datos en Hyperterminal resultasen claros y legibles.

PRESUPUESTO

COSTES DE MATERIALES

PCB

Placa de circuito impreso (x2).....36 €

Resistencias SMD.....10 valores distintos x 25(unidades cada valor) x 3,38 € = 33,8 €

Condensadores SMD.....2 x 25 (unidades) x 0,061 € = 0,122 €

Regulador lineal de tensión LM1117 (x5).....0,664 €

Regulador de tensión conmutado LM350(x5).....13,09 €

Circuito integrado MCP2551 (x2).....0,38 €

Clema de 2 conexiones (x5)..... 0,48 €

Subtotal 1……..……..84,53 €

MICROSTICK

Microstick.....18,70 €

Subtotal 2……..……..18,70 €

XBEE PRO 868 DEVELOPMENT KIT

Xbee Pro 868 Development Kit.....210,66 €

Subtotal 3…….…….…….210,66 €

Subtotal 1……...……..84,53 €

Subtotal 2……...……..18,70 €

Subtotal 3…….…….…….210,66 €

Subtotal Materiales.....……..... 313,89 €

COSTES DE INGENIERÍA

Duración del proyecto: 6 meses.

Horas de trabajo semanales: 40 horas.

Coste por hora de ingeniería: 30 euros.

Fases de desarrollo del proyecto:

- Estudio inicial: 80 horas (Incluye el estudio de la documentación y la búsqueda de una solución inicial).
- Desarrollo del software: 300 horas (Codificación de las aplicaciones LabVIEW y firmware del microprocesador).
- Desarrollo del hardware: 300 horas (Realización de los prototipos de las placas de circuito impreso).
- Fase de pruebas del sistema: 280 horas (Incluye las pruebas del sistema y el rediseño de los prototipos y la depuración del software).

Horas totales: 960 horas.

Subtotal Costes de Ingeniería…….....…28800 €

TOTAL.....29113,89 €

REFERENCIAS

[1] http://www.electriauto.com/electronica/can-bus/el-can-bus-de-datos/

[2] Datasheet de Microchip. Sección 21, Enhanced Controller Area Network.
http://ww1.microchip.com/downloads/en/DeviceDoc/70226C.pdf

[3] 1991, Robert Bosch GmbH, Postfach 30 02 40, D-70442 Stuttgart.
http://www.bosch-semiconductors.de/media/pdf_1/canliteratur/can2spec.pdf

[4] Function Manual MS3 Sport.

http://www.boschmotorsport.de/content/language2/html/3589.htm#a_0DEB5A9F763846A7BC0B325307803A8C

[5] Manual microprocesador PIC24HJ64GP502.

www.microchip.com/downloads/en/DeviceDoc/70293G.pdf

[6] Datasheet regulador de tensión LM350.

http://www.fairchildsemi.com/ds/LM/LM350.pdf

[7] Product Manual (Xbee Pro 868 RF Modules)

http://ftp1.digi.com/support/documentation/90001020_C.pdf

BIBLIOGRAFÍA

Microchip

Manual de usuario de MPLAB IDE

(1)http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB_User_Guide_51519c.pdf

Manual Microprocesador

(2) http://ww1.microchip.com/downloads/en/DeviceDoc/70293G.pdf

Seccion I/O Ports

(3) http://ww1.microchip.com/downloads/en/DeviceDoc/70193D.pdf

Seccion DMA

(4) http://ww1.microchip.com/downloads/en/DeviceDoc/70215C.pdf

Seccion ECAN Module

(5) http://ww1.microchip.com/downloads/en/DeviceDoc/70185C.pdf

Seccion Oscilador

(6) http://ww1.microchip.com/downloads/en/DeviceDoc/70216D.pdf

Seccion UART

(7) http://ww1.microchip.com/downloads/en/DeviceDoc/70188E.pdf

MCP2551

(8) http://ww1.microchip.com/downloads/en/DeviceDoc/21667E.pdf

Digi

Xbee Pro 868 Dev Kit

(9) http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-pro-868#docs

(10) http://ftp1.digi.com/support/documentation/90001003_A.pdf

(11) http://ftp1.digi.com/support/documentation/90002155_A.pdf

Otros

Especificación CAN

(12) http://www.semiconductors.bosch.de/pdf/can2spec.pdf

Regulador lineal

(13) http://www.national.com/ds/LM/LM1117.pdf

MAX232 Multichannel RS232 Drivers/Receivers

(14) http://www.datasheetcatalog.org/datasheet/maxim/MAX220-MAX249.pdf

Anexo A

MANUAL DE REFERENCIA

Contenido:

1. Introducción
2. Programación del microprocesador
3. Conexión de las PCBs
4. Configuración de los chips de radio
5. Conexión del receptor al PC de monitorización y LabVIEW

A.1 Introducción

El sistema desarrollado en este Proyecto Fin de Carrera es un sistema de telemetría que se encarga de transmitir, a través de una antena de radiofrecuencia, el estado de un vehículo de carreras, que es recogido desde el bus CAN del mismo. El sistema consta de tres placas de circuito impreso (dos componen el emisor y la restante el receptor) y una aplicación software en LabVIEW. En el siguiente manual se describen los pasos a seguir para poner en funcionamiento el sistema para poder monitorizar los datos.

A.2 Programación del microprocesador

Para poner en funcionamiento el sistema, lo primero que hay que hacer es programar el microprocesador que recibirá los datos del bus CAN de la centralita y los transmitirá al emisor de radio. Para ello, hay que seguir una serie de pasos que se detallan a continuación:

- Conectar el microprocesador en la placa de desarrollo Microstick.

- Configurar MPLAB.

Conectar el microprocesador a la placa Microstick

Para programar el micro con el código de la aplicación se debe pinchar este en la tarjeta de desarrollo Microstcik. La tarjeta se conecta al ordenador mediante un cable USB a mini-USB

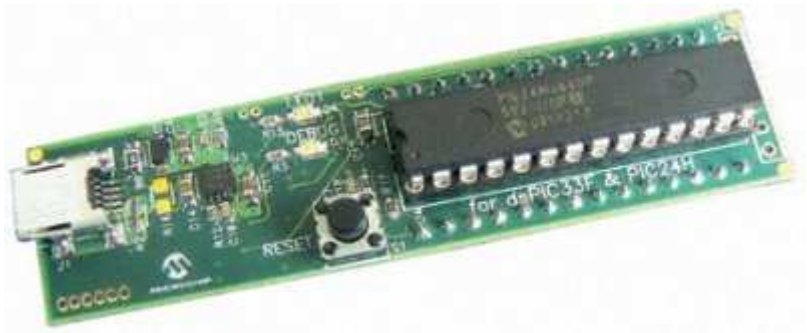


Figura a.1 – Tarjeta de desarrollo Microstick

Configuración de MPLAB

Una vez se tiene conectada la tarjeta Microstick es necesario realizar una serie de acciones para programar el microprocesador. La primera es abrir MPLAB. Una vez abierto MPLAB, se debe seleccionar el programador adecuado. Si se trabaja con la tarjeta Microstick hay que pinchar sobre la opción Select Programmer del menú Programmer y seleccionar la opción Starter kits y si se trabaja con la tarjeta microstick II seleccionar la opción Starter Kit on Board dentro del mismo menú.

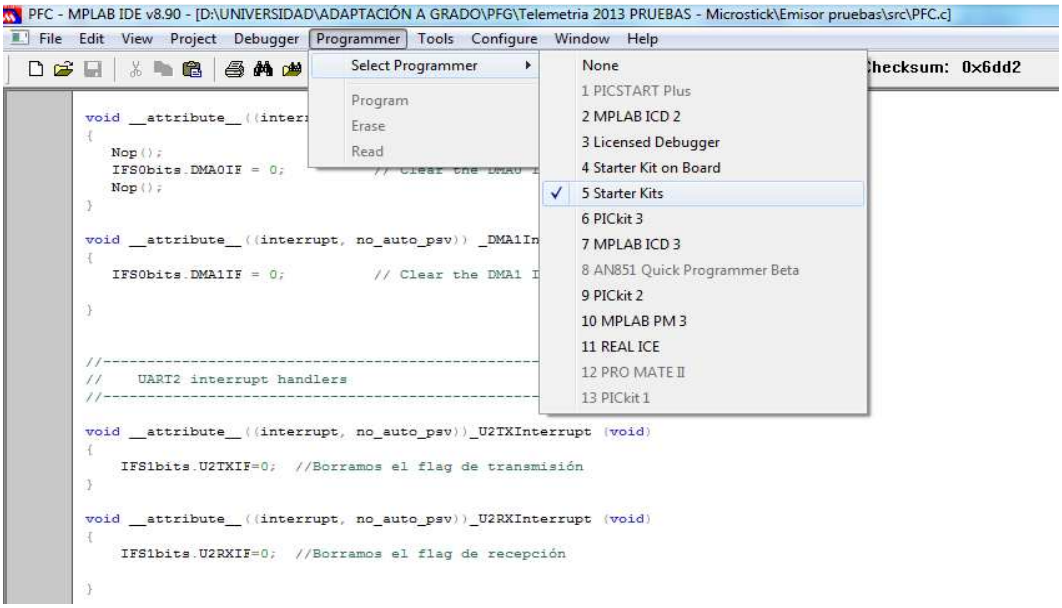


Figura a.2 – Selección del programador

Por último, seleccionar la opción Program del menú Programmer.

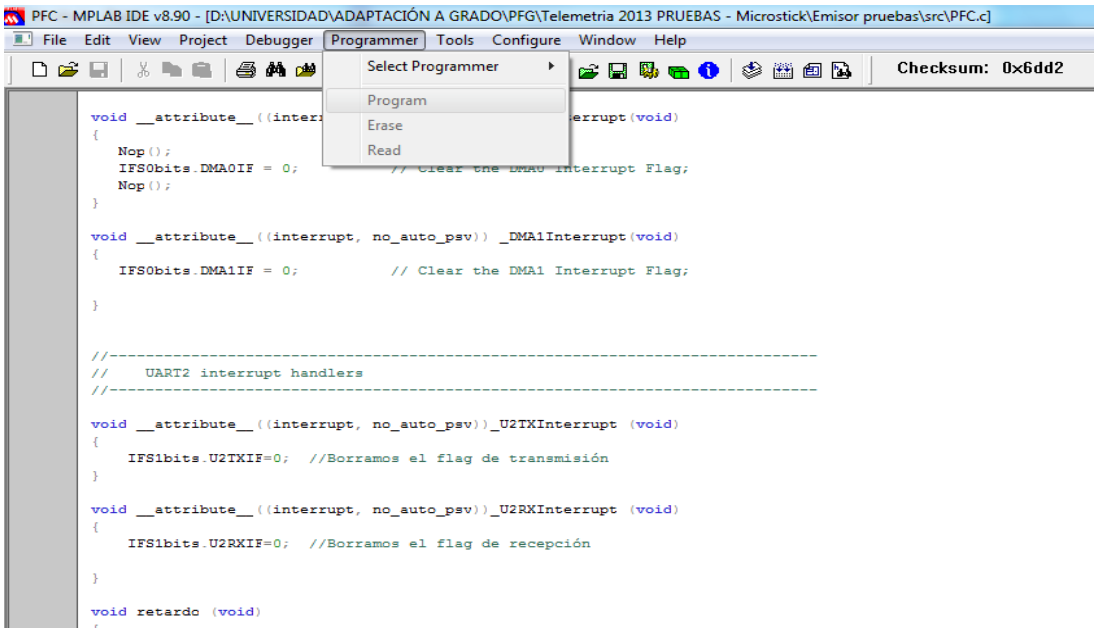


Figura a.3 – Opción de programar el microprocesador

A.3 Conexión de las PCBs

Una vez programado el microprocesador, se puede proceder a conectar las placas. Las dos PCBs que componen el emisor del sistema se interconectan a través de dos clemas. Enumerando las conexiones de abajo hacia arriba, se deben conectar los pines 1 (0V), 3 (5V), 4 (Linea de Tx serie) y 7 (3,3V)

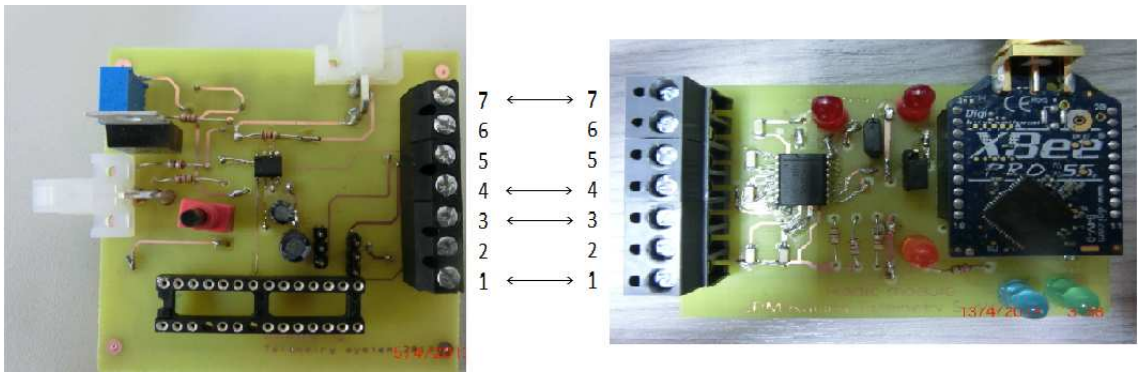


Figura a.4 – Interconexión de las PCBs

La placa que contiene al microprocesador (a la izquierda en la imagen anterior) recibe alimentación de la batería del coche y es la que proporciona alimentación a la otra placa (derecha en la imagen) a través de la clema. También la placa del microprocesador es la que va conectada al bus CAN de la ECU. A continuación se detalla cual es cada conector.

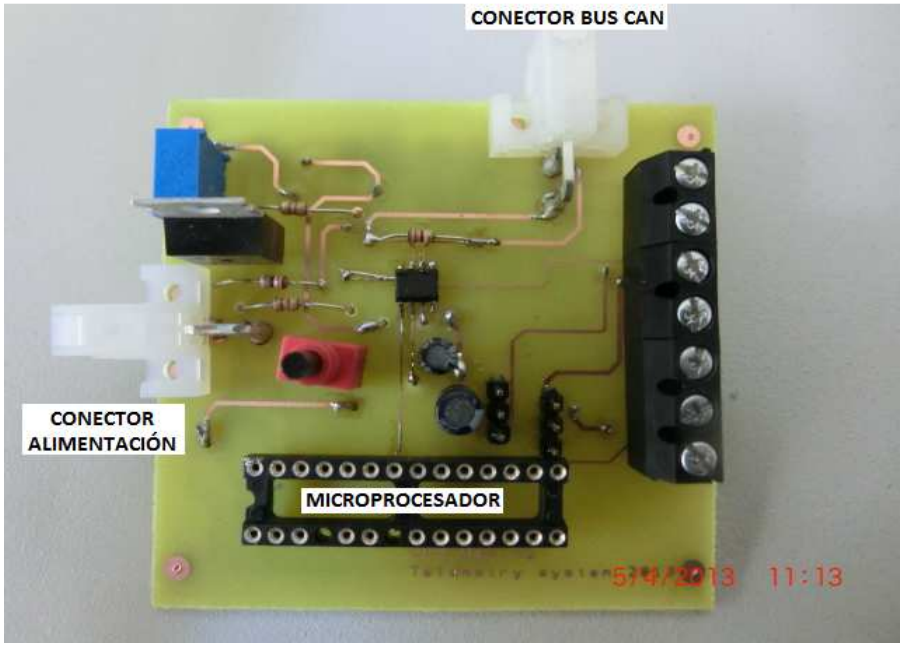


Figura a.5 – Conexión de la PCB del microprocesador

El bus CAN y la batería del vehículo deben conectarse como se indica a continuación:

Conector Alimentación

Pin superior → Masa Batería

Pin inferior → +12V

Conector bus CAN

Pin superior → CAN High

Pin inferior → CAN Low

A.4 Configuración de los chips de radio

Para configurar los chips de radio el fabricante proporciona un par de documentos donde explica todos pasos necesarios para poder empezar a trabajar con los módulos, desde la instalación del software X-CTU que permite comunicarse con los chips, la instalación de los drivers hasta la configuración básica para poder transmitir datos. Dichos manuales se pueden encontrar en:

http://ftp1.digi.com/support/documentation/90001003_A.pdf

http://ftp1.digi.com/support/documentation/90002155_A.pdf

A.5 Conexión del receptor al PC de monitorización y LabVIEW

Una vez configurados los módulos de radio y conectados cada uno en su PCB, el último paso para poder empezar a recibir datos es ejecutar la aplicación programada en LabVIEW. Para ello abrir el fichero llamado **Telemetría combustión**. En LabVIEW se tienen dos pantallas, una llamada Panel Frontal, en la que se visualizan los datos y otra llamada Diagrama de Bloques, donde se encuentra el código del programa.

En el Diagrama de Bloques se debe seleccionar el número de puerto por el que se van a recibir los datos (se debe seleccionar el mismo número de puerto que aparecía en el software X-CTU al configurar el chip de radio que se quiere que actúe como receptor).

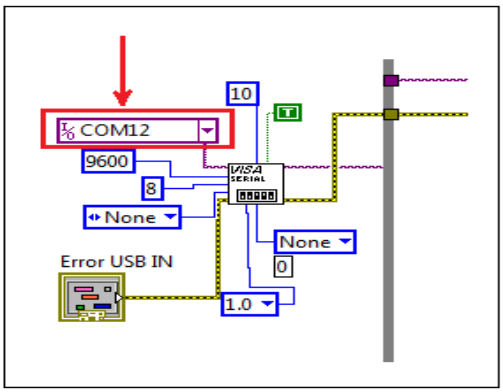


Figura a.6 – Selección del puerto de comunicaciones en LabVIEW

Cuando se decide finalizar la aplicación, todos los datos que se han ido tomando se guardan en un fichero. El usuario puede seleccionar en el Diagrama de Bloques y antes de iniciar la toma de datos, la ruta donde se almacenará el fichero y el nombre del mismo.

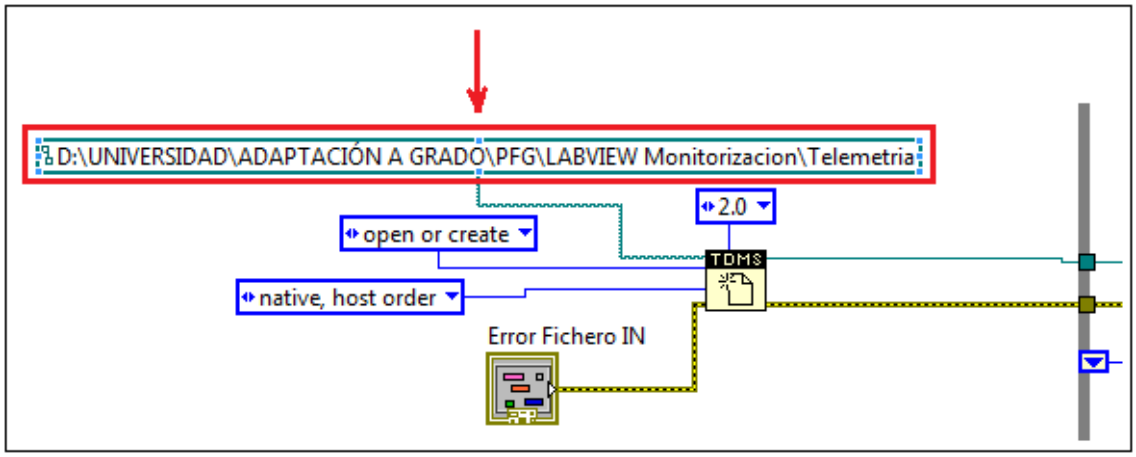


Figura a.7 – Elección del nombre y el destino para el fichero de datos

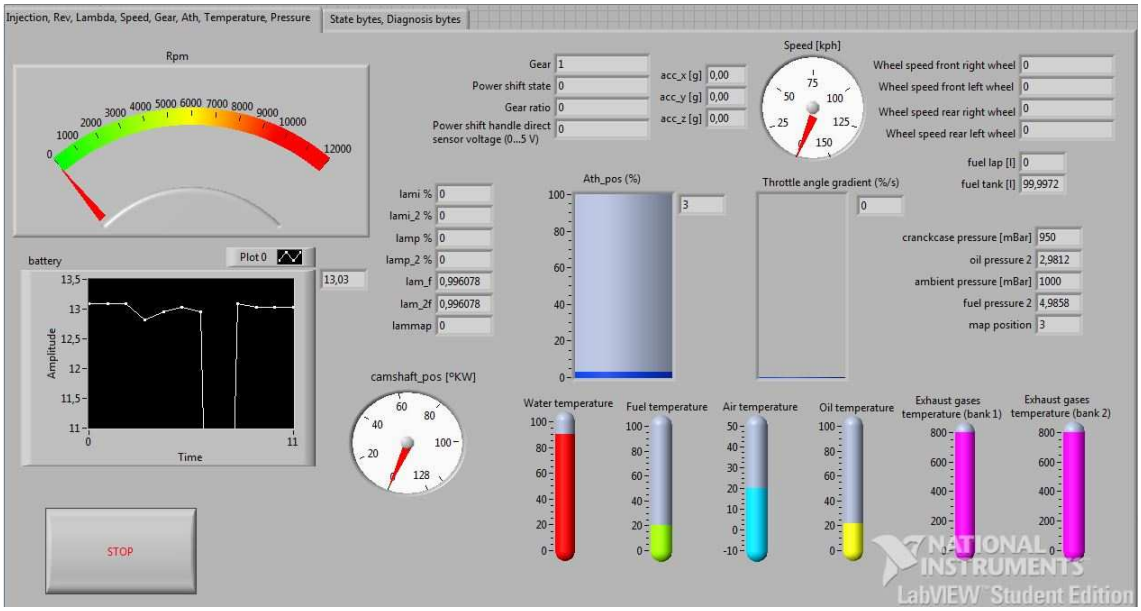


Figura a.8 – Panel Frontal de la aplicación LabVIEW

Por último, una vez grabados los datos, se puede leer el fichero generado con otra pequeña aplicación en LabVIEW, en la cual se pueden elegir que variables se quieren visualizar, de manera que estas se representan conjuntamente en forma de gráficas. El fichero ejecutable para esta aplicación se llama **Leer Fichero Telemetría.**

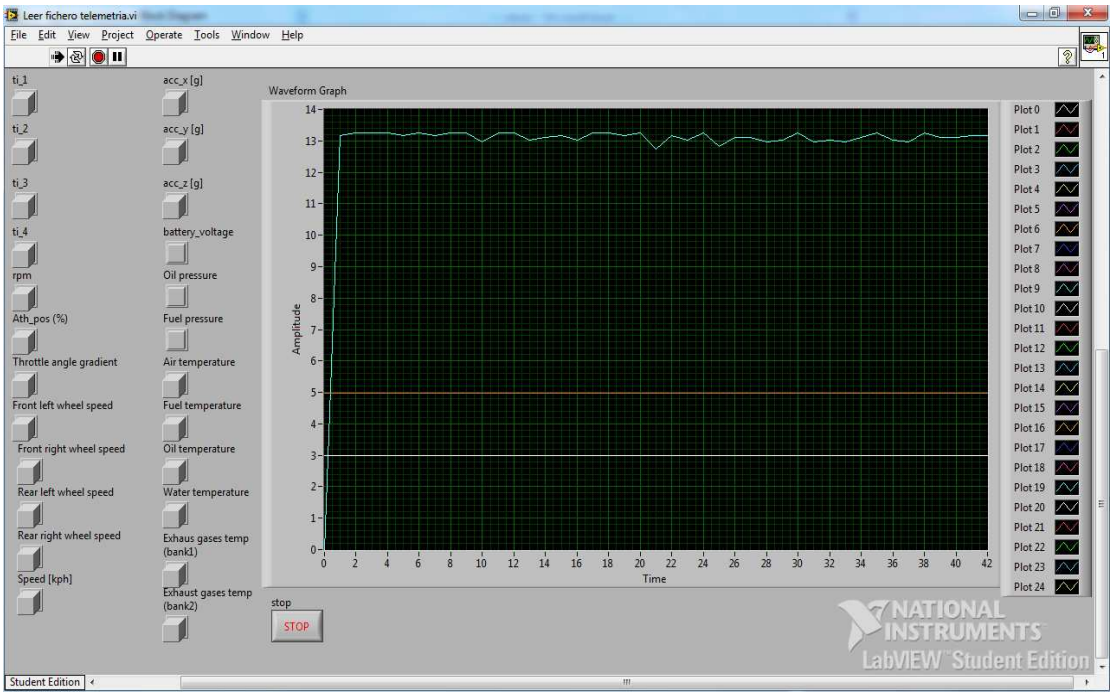


Figura a.9 – Representación gráfica de los datos adquiridos

Anexo B

PLANOS

1. Esquemático PCB microprocesador

   1.1Bloque alimentación

   1.2 Microprocesador

   1.3 Comunicaciones serie

   1.4 Comunicaciones CAN

2. Esquemático PCB chip de radio

1. Esquemático PCB microprocesador

1.1Bloque alimentación

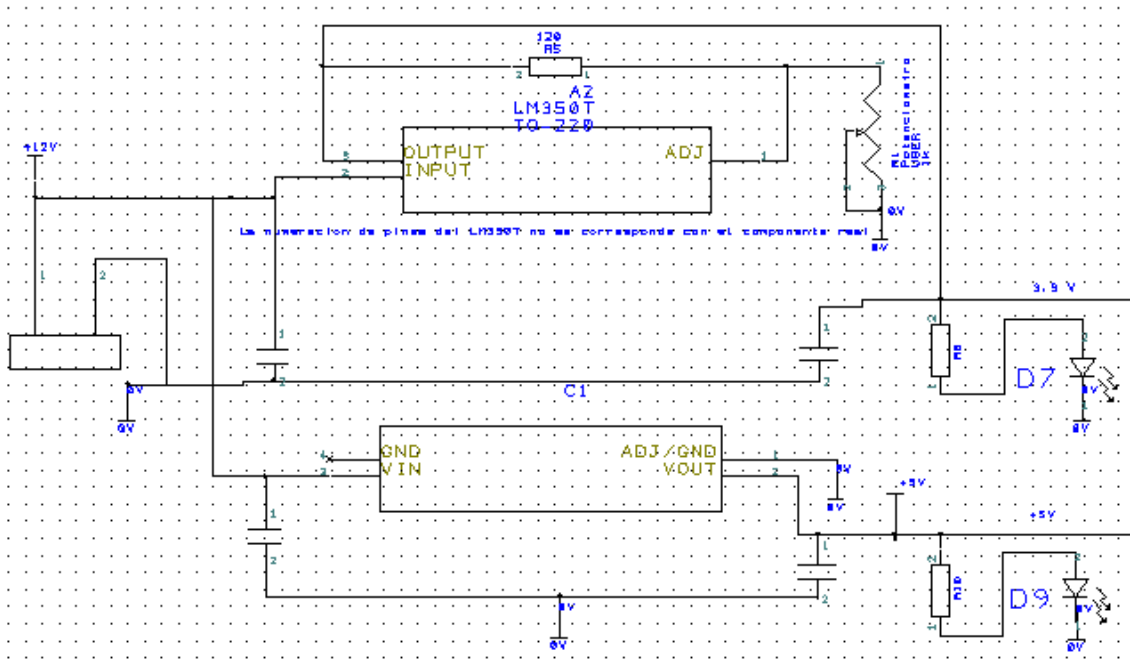


Figura b.1 – Bloque de alimentación de las PCBs

1.2 Microprocesador

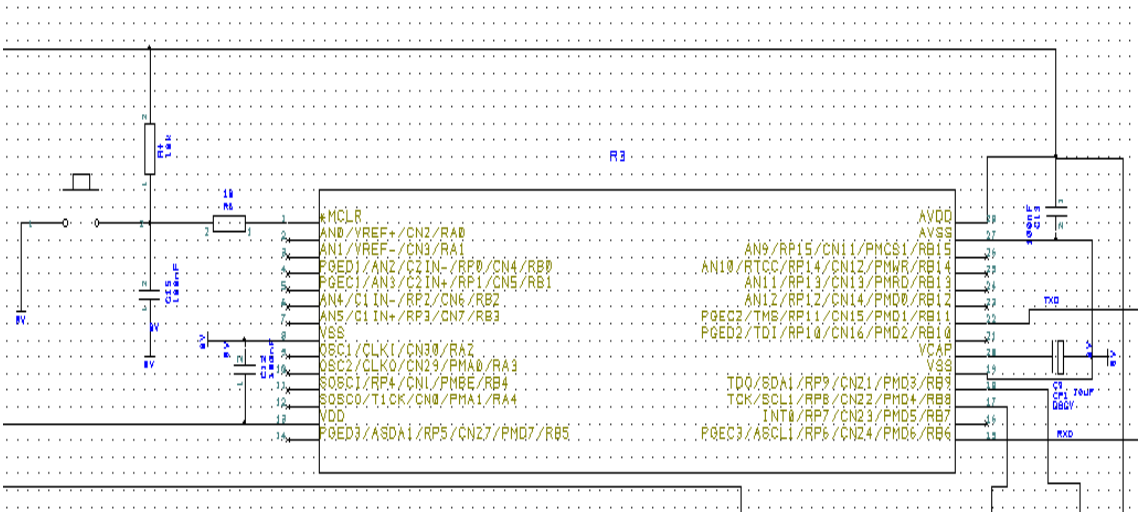


Figura b.2 – Esquemático del microprocesador

1.3 Comunicaciones serie

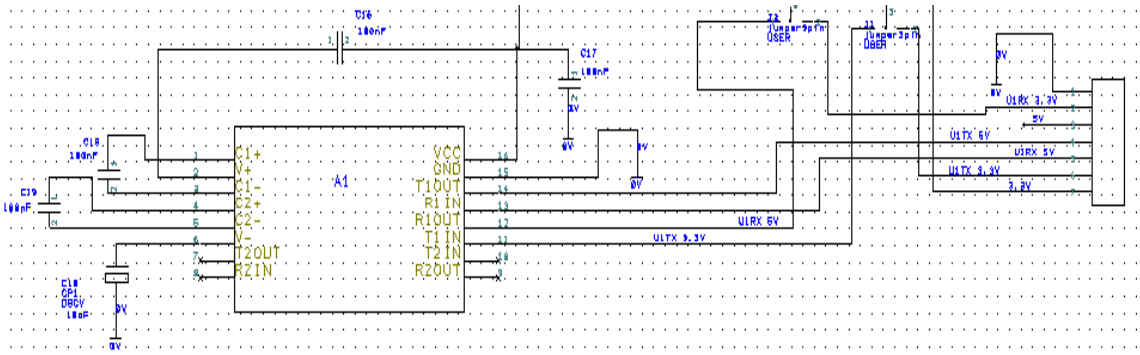


Figura b.3 – Comunicaciones serie

1.4 Comunicaciones CAN

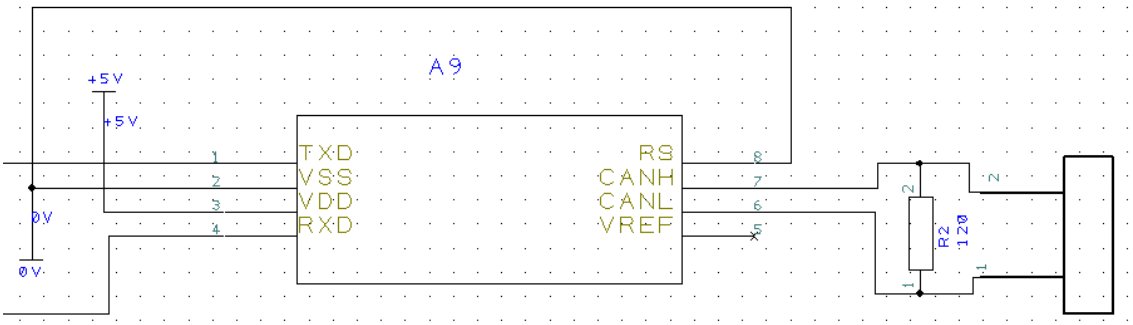


Figura b.4 – Comunicaciones CAN

2. Esquemático PCB chip de radio

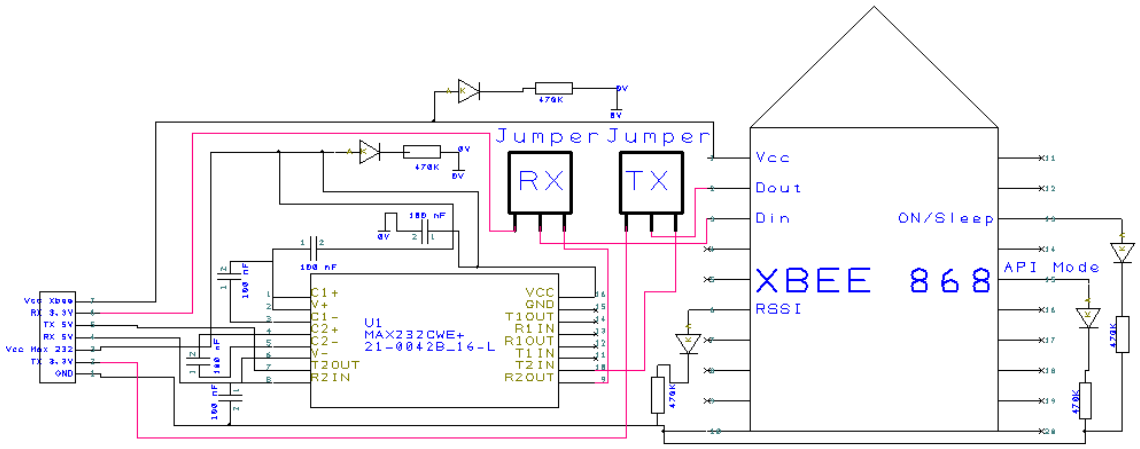


Figura b.5 – Esquemático de la PCB del módulo de radio